UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería informática

TRABAJO FIN DE GRADO

Desarrollo de un Sistema de Gestión de Reservas basado en Modelado de Procesos

Víctor Ramírez López
Tutor: Alfonso Díez Rubio
Ponente: Fernando Díez Rubio

Julio 2015

Resumen

Hoy en día la ingeniería dirigida por modelos es un ámbito que aún no se conoce mucho y queda mucho por descubrir. Actualmente las empresas que se dedican al desarrollo dirigido por modelos están en pleno auge ya que en los últimos años se empiezan a conseguir los resultados esperados y se ha avanzado mucho en el desarrollo de software utilizando esta metodología.

El objetivo de esta metodología es desarrollar software sin escribir líneas de código.

En este trabajo de fin de grado se desarrollan tanto una aplicación web y como una aplicación móvil utilizando únicamente la ingeniería dirigida por modelos.

La aplicación desarrollada da soporte para la gestión de reservas que los usuarios realizan sobre las distintas salas que la empresa posea.

En este proyecto la aplicación móvil solamente ha necesitado de unas 50 líneas de código, por lo que el objetivo de conseguir software totalmente funcional sin escribir ni una sola línea de código podría decirse que está cerca.

Palabras clave

MDE, reserva, sala, localización, reserva recurrente

Abstract

Today the model driven engineering is an area that not much is known yet and much is still unknown. Companies currently engaged in model-driven development are booming because in recent years are beginning to achieve the expected results and progress has been made in developing software using this methodology.

The objective of this methodology is to develop software without writing lines of code.

In this final degree project develop both a Web application and a mobile application using only model-driven engineering.

The application developed supports the management of reserves that users make on the various rooms that the company owns.

In this project the mobile application only needed about 50 lines of code, so in order to get fully functional software without writing a single line of code could be said to be close.

Keywords

MDE, reserve, room, location, recurrent reserve

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

La ingeniería dirigida por modelos (MDE de sus siglas en inglés “Model Driven Engineering”) es una metodología de desarrollo de software que pretende desarrollar software de gran calidad y de una forma más rápida a la habitual.
La MDE se centra en elevar el nivel de abstracción, por lo que en vez de construir software mediante líneas de código puramente, se crean mediante modelos que evitarán grandes cantidades de líneas de código.
Un modelo es una representación simplificada o parcial de la realidad con el fin de realizar una tarea.
La MDE contiene lo que se define como Desarrollo Dirigido por Modelos (MDD en inglés Model Driven Developement), que entre sus características más importantes destaca el aumento de la productividad, aumenta la calidad del software, la evolución y mantenimiento del software se realiza de una manera más sencilla…
Los Modelos normalmente se transforman en otros modelos, lo que se conoce como M2M (Model to Model) y finalmente, la última transformación es la que se conoce como M2T (Model to Text).
En este trabajo de fin de grado se desarrollará una aplicación web y una aplicación móvil (IOS y Android) utilizando MDE, en concreto definiendo dentro de un modelo todos los conceptos que este debe tener para que una vez definido, sea capaz de generar las dos aplicaciones deseadas.
Existen distintas arquitecturas para aplicar este tipo de ingeniería, en este proyecto se empleará la utilizada en la empresa B2T la cual utiliza la herramienta SCooP para diseñar los modelos.

1.2 Objetivos

El objetivo principal de este proyecto es desarrollar una aplicación Web y una aplicación móvil mediante la ingeniería de modelos que sea capaz de gestionar las reservas que se realizan en el sistema.

Otro de los objetivos más importantes es conseguir que la aplicación funcione tanto en dispositivos Android como en IOS.

Interfaz intuitiva: La interfaz gráfica de la aplicación tiene que ser intuitiva y fácil de usar para facilitar al usuario la navegación por el sistema.

Rapidez en las consultas de páginas: Con el fin de evitar que el usuario tenga largas esperas en algunas pantallas, la navegación por el sistema tiene que ser fluida.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **1-Introducción:** En esta sección se indicará la motivación y objetivos del proyecto.

- **2-**Teconlogías a desarrollar: En esta sección se comentan brevemente las aplicaciones y herramientas que se han utilizado para llevar a cabo el desarrollo del proyecto.
- **3-**Diseño de la aplicación: Se cuenta la fase previa al desarrollo, es decir, los requisitos de la aplicación y lo que se espera desarrollar.
- **4-**Desarrollo: Se definen los distintos módulos que va a tener el proyecto así como algunas imágenes de la aplicación final.
- **5-**Pruebas: Se detallan algunas de las pruebas realizadas al sistema desarrollado.
- **6-Conclusiones:** En este punto se extraen las conclusiones que se han sacado al desarrollar el proyecto y el trabajo futuro que se va a realizar en él.

2 Tecnologías a utilizar

En el TFG desarrollado se utilizará la tecnología que se utiliza y, en muchos casos ha sido desarrollada, por la empresa B2TConcept. Esta tecnología está orientada a MDE (Mode Driven Engineering) y cuenta con la arquitectura definida en la siguiente figura.

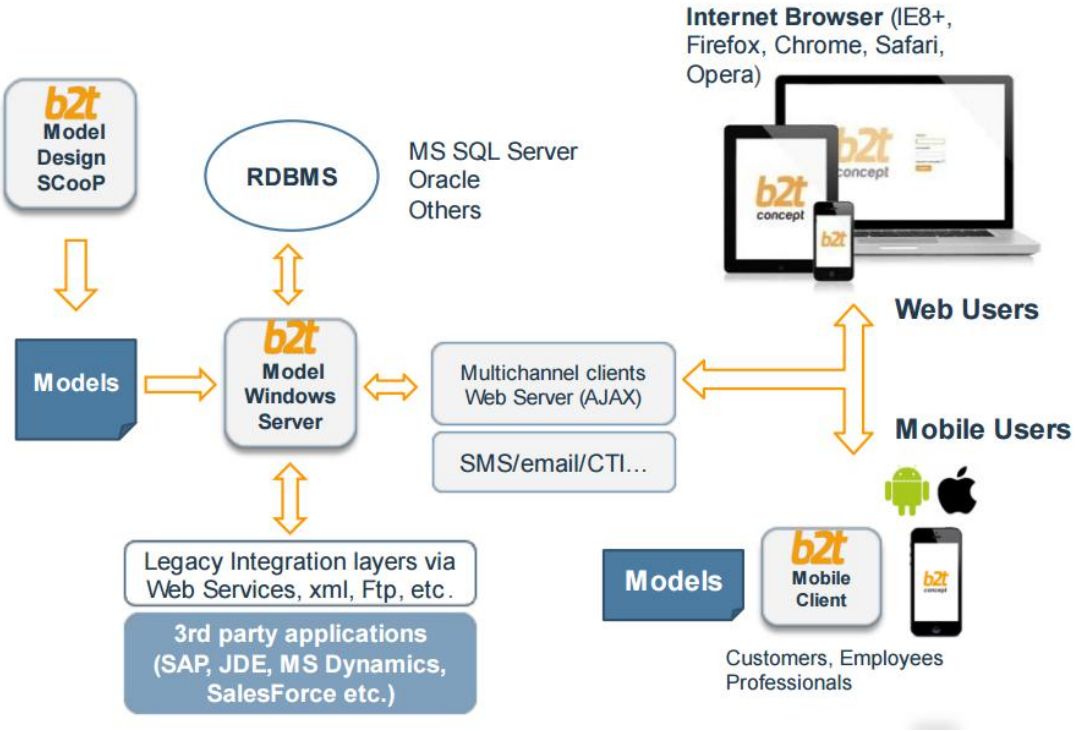


Figura 2-1: Arquitectura B2T

A continuación se describen cada uno de los elementos que se muestran en la figura y algunos más que, aunque no aparecen en la figura se han utilizado para desarrollar el proyecto.

JCOR: Lenguaje de programación interpretado y orientado a objetos desarrollado por la compañía B2TConcept. Técnicamente es un lenguaje muy similar a JavaScript, por lo que tienen muchas características en común. En la figura estaría representado en los “Models” generados por SCooP.

Javascript: Lenguaje de programación interpretado orientado a objetos, basado en prototipos, imperativo débilmente tipado y dinámico. Se ha utilizado en el desarrollo de la aplicación móvil.

DEVR: Entorno de desarrollo propio de la compañía B2TConcept. Tiene un motor que interpreta el código desarrollado en JCOR. En el entorno se puede ver el fichero de configuración inicial de la aplicación (fichero .ini), las Querys ejecutadas por el sistema a la base de datos y todo el flujo del sistema. También dispone de un depurador de JCOR.

Subversion (SVN): Herramienta de control de versiones Open Source basada en un repositorio. En este proyecto será el encargado de manejar las versiones entre el código generado por SCooP y el código desarrollado por el usuario en JCOR, es decir, relaciona SCooP con el código desarrollado por el usuario.

SCOOP: Establece un marco de referencia, un meta-modelo sobre el cual se hacen definiciones de funcionamiento del modelo de negocio que se quiere abarcar. En SCOOP es donde se describen los objetos del negocio que vamos a manipular, sus características y las reglas de negocio que vamos a aplicar sobre ellos. Esto quiere decir, el desarrollador introduce en SCooP el modelo de negocio adecuado y SCooP lo transforma en una aplicación de negocio eficiente. El diseño de modelos está en la nube y gracias a esta herramienta se pueden definir procesos muy detalladamente, requisitos de negocio, interfaces de usuario, menús de la aplicación, acciones de los usuarios, actores, la seguridad de los roles de usuario y mucho más.
Gracias a SCooP podemos desarrollar tanto aplicaciones web como aplicaciones para dispositivos móviles que utilicen la misma lógica.
SCooP nos proporciona unos ficheros en lenguaje JCOR de forma automática, los cuales dividimos en ficheros de clases y ficheros de funciones. Los ficheros de clases no se pueden modificar, por el contrario los ficheros de funciones son los que el desarrollador debe definir para posteriormente sincronizarlos con SCooP de nuevo mediante SVN.
Dentro de la arquitectura de B2T, SCooP es el Diseñador de modelos.

BPM: Proceso de negocio que tiene asociados distintos eventos. Los eventos son gestionados por el BPM y en función del flujo definido se toman unas decisiones u otras. Los BPM pueden ser de muchos tipos, pero en este proyecto solo se utilizarán los que manejan los estados de los objetos. Están definidos en una librería que cuenta con todas las funciones necesarias para manejar los distintos BPM.

E3: Es el motor de gestión de modelos de negocio generados por SCOOP. Es un sistema generalizado que permite ejecutar cualquier tipo de modelo de negocio escrito con metodología SCOOP. Al ejecutar los modelos producidos por SCOOP, produce soluciones de negocio basados en ellos. En E3 se puede ejecutar cualquier tipo de modelo empresarial, desde el modelo que aplica un hospital a sus pacientes o a su gestión de medicamentes, el modelo que aplica una empresa comercial a sus clientes o a sus almacenes…En la arquitectura B2T E3 se corresponde a Model Windows Server

SABLE: Las interfaces de usuario que nacen a partir de E3 y SCOOP se gestionan mediante el sistema SABLE. Es un generador automático de interfaces de usuario que se basa en las definiciones que hacen E3 y SCOOP para poder generar de forma dinámica interfaces de usuario para los clientes. Está desarrollado con tecnología Cross Browser y permite el desarrollo de cualquier tipo de interfaz. El sistema admite que la interfaz Sable sea total o parcialmente sustituida por interfaces realizadas a medida, lo cual dota de una gran flexibilidad a todo el modelo de gestión.
Con todo esto generamos soluciones de negocio muy avanzadas. Desarrollamos un modelo de negocio en SCOOP, se lo aplicamos a un modelo E3 de ejecución,

pasa por la interfaz Sable para generar las interfaces de usuario, y el cliente recibe la aplicación que interpreta las reglas de negocio descritas en SCOOP.
En la arquitectura B2T SABLE se encuentra en Multichannel Clients Web Server

MS SQL Server 2008: Sistema de gestión de bases de datos relacionales desarrollado por Microsoft.

Cascading Style Sheets(CSS): Lenguaje que sirve para definir la presentación de documentos escritos en HTML o XML.

HTML5: Lenguaje de marcas que se utiliza para definir páginas web. Quinta revisión del lenguaje HTML. En esta quinta revisión se han añadido y corregido nuevas características, entre las que cabe destacar el soporte de contenido multimedia sin tener que recurrir a plugins y APIs.

3 Diseño

Los requisitos del sistema se componen de un conjunto de requisitos iniciales especificados por el cliente y que posteriormente fueron modificados a medida que se iba viendo el resultado final del desarrollo. No se distingue en requisitos funcionales y no funcionales ya que todos los requisitos descritos se han considerado funcionales.

3.1 Requisitos Iniciales

El sistema, en su propuesta inicial estaba definido de acuerdo a una serie de requisitos por el cliente contratante del proyecto. A continuación describo algunos de estos requisitos agrupados por módulos del sistema.

3.1.1 Localizaciones

- Las localizaciones solamente dispondrán del campo Nombre.
- Si se desea borrar una localización, no puede tener ninguna reserva en la base de datos asociada.

3.1.2 Registro de salas

Las salas pueden ser dadas de alta, modificarlas, activar/desactivar y borrar.

- El nombre de la sala
- Localización
- La capacidad de la misma
- El nº de puntos de Red
- Si dispone de Proyector
- Si dispone de Video-Conferencia.
- Otras características adicionales.
- Todos los usuarios podrán consultar las salas.
- Sólo los administradores podrán modificar los datos de salas.
- Borrar: Si una sala tiene reservas en la BBDD se hará borrado lógico, si no tiene reservas se borrará de base de datos

3.1.3 Búsqueda de salas

Existirá un buscador de salas, en el que se podrá filtrar por los siguientes campos.

- Fecha (Desde - Hasta) – obligatorio
- Hora (Inicio - Fin)
- Duración
- Capacidad
- El nº de puntos de Red
- Si dispone de Proyector
- Si dispone de Video-Conferencia.
- Localización
- Sala
- Los resultados de la búsqueda se presentarán mediante un listado con las salas disponibles que cumplan los criterios de búsqueda

- Al seleccionar una sala, se abrirá la página de alta de reserva

3.1.4 Cancelación de reservas

- El usuario que ha reservado una sala podrá cambiar o cancelar la reserva hasta 15 minutos antes del inicio de la reserva.

3.1.5 Reservas

Para dar de alta reservas son necesarios los siguientes campos:

- Título de la reserva.
- Fecha de la reserva (Día/Mes/Año) y hora de inicio y fin de la misma.
- Todas las reservas tienen que tener una sala indicada.
- Si una reserva se hace sobre una sala con responsable, se creará un BPM que deberá administrar el responsable de la sala.
- Se pueden solicitar reservas en nombre de otra persona (Solo Admin)

3.1.6 Reservas recurrentes

Para dar de alta reservas recurrentes son necesarios los siguientes campos:

- Título
- Día de la semana: L - V
- Cadencia: Cada semana, 1º semana del mes, 2º semana del mes, 3º semana del mes, última semana del mes
- Sala
- El sistema generará las reservas correspondientes. Si no se puede generar alguna reserva, el sistema enviará un email al solicitante
- Una reserva recurrente se puede modificar - solo el título
- Una reserva recurrente se puede dar de baja - se borrarán las reservas pendientes
- Si la sala tiene responsable, se creará un BPME que debe ser gestionado por el responsable de la misma. Si la reserva recurrente es aceptada se crearán las reservas individuales sin necesidad de ser aprobadas.
- Una reserva generada se podrá borrar pero no modificar
- En página de alta mostrar listado de conflictos con las reservas individuales existentes en el horario indicado (independientemente se envía mail después)
- La APP para móvil no dispondrá de reservas recurrentes.

3.1.7 Consulta de salas

- En la consulta de una sala se deben mostrar todos los datos de la misma.
- Si la sala tiene reservas, tanto individuales como recurrentes, se deben mostrar (Solo reservas futuras).

3.1.8 Permisos de los usuarios

Solo existirán 2 tipos de usuario, administrador y usuario normal.

- Admin Salas. Tendrá permiso de modificar y borrar Reservas, dar de alta reservas en nombre de otra persona, gestionará las localizaciones y salas del sistema así como las solicitudes de acceso.
- Rol Usuario salas - puede modificar y borrar solo sus propias reservas.

3.1.9 Requisitos APP Móvil

- Pantalla inicial para seleccionar “Mis reservas” o “Buscar sala”
- En “Mis reservas” mostrar lista con todas las reservas realizadas por el usuario (Solo reservas individuales)
- Las reservas se pueden consultar, editar y anular.
- En caso de buscar sala, se puede realizar mediante 2 opciones, “Filtros” y “Salas”. Desde la opción de filtros el buscador será general en una localización indicada con los parámetros indicados(Los mismos que en versión web). Si buscamos por “Salas”, tendremos que indicar la sala de la que queremos ver sus reservas.
- Las opciones de filtrado son 2. “Ver ocupación” y “Ver disponibilidad”.
- Buscando con “Ver ocupación” mostrará las reservas disponibles y las reservas realizadas en el sistema, es decir, toda la información de la sala.
- Buscando por “Ver disponibilidad” solamente mostrará las reservas disponibles.
- En ambas pantallas se debe mostrar la información de la sala y la posibilidad de ver la disponibilidad un día anterior o posterior.
- Mostrar mensajes de confirmación tanto al reservar como al borrar reservas.

3.2 Requisitos durante el desarrollo

Durante el desarrollo de la aplicación se han modificado algunos requisitos descritos anteriormente y se han definido otros requisitos completamente nuevos.

3.2.1 Cambios en Reservas

- En Nueva Reserva: si no se selecciona sala debe mostrarse un HTML análogo al de Buscar Sala (en lugar de 'Reservar 10:00 - 17:00' debe salir 'Libre 10:00 - 17:00' inhibido (sin link con proc)
- Si se viene del buscador debe tenerse en cuenta la duración indicada (WEB y APP)
- En el HTML debe mostrarse un texto encima: sala elegida, horario, etc
- El listado de otras reservas debe mostrarse en otra solapa - solo cuando procede (reserva normal y recur)
- El HTML de reserva normal con sala seleccionada debe ser parecido al de buscar sala:

3.2.2 Reserva Recurrente:

- Añadir Fecha inicio y fin opcionales
- Arreglar HTML de reserva recursiva parecido al de reserva.
- En el HTML de reserva recurrente poner información al pasar por encima de una reserva parecido a Reserva - título, solicitante, vigencia
- No hay que mostrar reservas en el HTML que no coincidan con el periodo de vigencia de la nueva

3.2.3 Reserva / reserva recurrente para Sala con Aprobación:

- Mostrar listado de reservas / reserva recurrente Pend. Aprobación en el mismo horario. Anteriormente solo se mostraban las reservas aprobadas.

3.2.4 Reserva generada desde Recurrente:

- Al borrar - mensaje de confirmación: 'Esta reserva se ha generado a partir de una reserva recurrente. ¿Desea borrar esta reserva individual o la programación completa?' - botones 'Borrar reserva', 'Borrar programación', 'Cancelar'
- Mostrar mensaje al modificar una reserva / reserva recurrente Aprobada. Si sólo se cambia el título, la hora de inicio no difiere en más de una hora o la duración no difiere en más de media hora la reserva sigue aprobada, en caso contrario si la sala necesita aprobación - avisar al usuario y relanzar BPME.

3.2.5 Buscar sala (WEB)

Mostrar los resultados en la página de búsqueda mediante un HTML:
- Mostrar localización en el HTML

	08:00	09:00		
Guzmán el bueno (localización A)				
Sala A				
Sala B				
Nuevos ministerios (local. B)				
Sala A				

- Ordenar localizaciones y salas por orden alfabético.
- Por defecto debe estar marcado Sala dipsonible = si
- Añadir una leyenda con los colores que salen en el HTML (parecido a reserva recurrente.)

3.2.6 Buscar Sala (Móvil)

- Resultados: cambiar texto Libre por descripción de la sala: 9pax, Proyector, Video-Conf., (salto de línea) texto otras características
- Pestañas: 'Filtros' y 'Salas'
- Localización - etiqueta inline
- Horario - sin etiqueta
- Duración - combo, etiqueta inline
- Capacidad - 'Asistentes'
- Proyector, Video - en misma línea
- 'Salas': todas las etiquetas inline

3.2.7 Cambios en buscador:

- Buscar por duración - hacer algoritmo en JCOR. Si se filtra por duración - filtrar libres solo si Ver disponibilidad
- Si fecha = Hoy tener en cuenta la hora actual al mostrar los resultados

- Ordenar resultados: Libre, Hora Inicio
- Duración (Móvil + WEB): Valores: 15min, 30min, 45 min, 1h, 1h 30min, 2h, 2h 30min,.....,11h 30min, Día entero

3.2.8 Consulta reserva (Móvil)

- Mostrar día de la semana junto a la fecha
- Mostrar estado de la reserva
- Editar - botón Guardar + mensaje confirmación

3.2.9 Consulta Reserva (WEB)

- En título de página poner Reserva + título reserva
- Cambiar 'Borrar' por 'Anular'
- Info sala - poner en un multiline debajo de combo sala

3.2.10 Cambios Salas

- Si está Activa - Modificar, Consultar, Desactivar (se permite si no existen reserva futuras – Mensaje confirmación – Marcar como inactiva)
- Si está Inactiva - Consultar, Activar, Borrar (Las salas pueden borrarse de BBDD solo si están inactivas y no existen reservas en BBDD)

3.2.11 Cambios generales

- En todos los combos Localización - Sala: si no se selecciona Localización deben salir todas las Salas ordenadas por orden alfabético
- En los HTML en la tabla hay que poner fondo blanco para que no salga el fondo naranja al pasar el ratón.
- Si se borra o da de baja una reserva o recurrente. con BPME activo - este debe cancelarse.
- Poner un panel HTML en pantalla de acceso con ayuda y links a los procedimientos de la izquierda

3.2.12 Registro de usuarios:

- Nombre y Apellidos, email, teléfono
- Se registra la solicitud (PDTE_APROB) y se envía mail de confirmación
- Se lanza BPM aprobación que puede ser de aprobado automático
- Va a haber una lista de mails (~500) que serán aprobados automáticamente
- Se genera usuario con login = mail, se genera contraseña y se envía mail al solicitante

3.2.13 Solicitudes de acceso

- Se creará un buscador de solicitudes de acceso que filtre por nombre, apellidos, email, teléfono, fecha de alta de la solicitud y estado de la misma
- En la consulta de cada solicitud se mostrará el estado de la misma y, en caso de tener un BPME activo, se mostrará y se podrán tomar decisiones al respecto.

3.3 Base de datos

La base de datos del proyecto se ha implementado utilizando Microsoft SQL Server 2008. Todas las tablas tienen un campo identificador (ID), el cual está identificado como NOMBRETABLA_ID. Como se puede ver a continuación, muchas tablas tienen en común algunos campos, como ALTA_USR, MOD_USR, ALTA_DT... Estos campos son para dejar registro en el sistema de quien ha realizado acciones sobre estos objetos. En la figura X se puede ver el diagrama ER de la BBDD implementada. Se omite la tabla Usuarios, una tabla del sistema que no ha sido desarrollada en este proyecto. Esta tabla cuenta con toda la información de los usuarios del sistema.

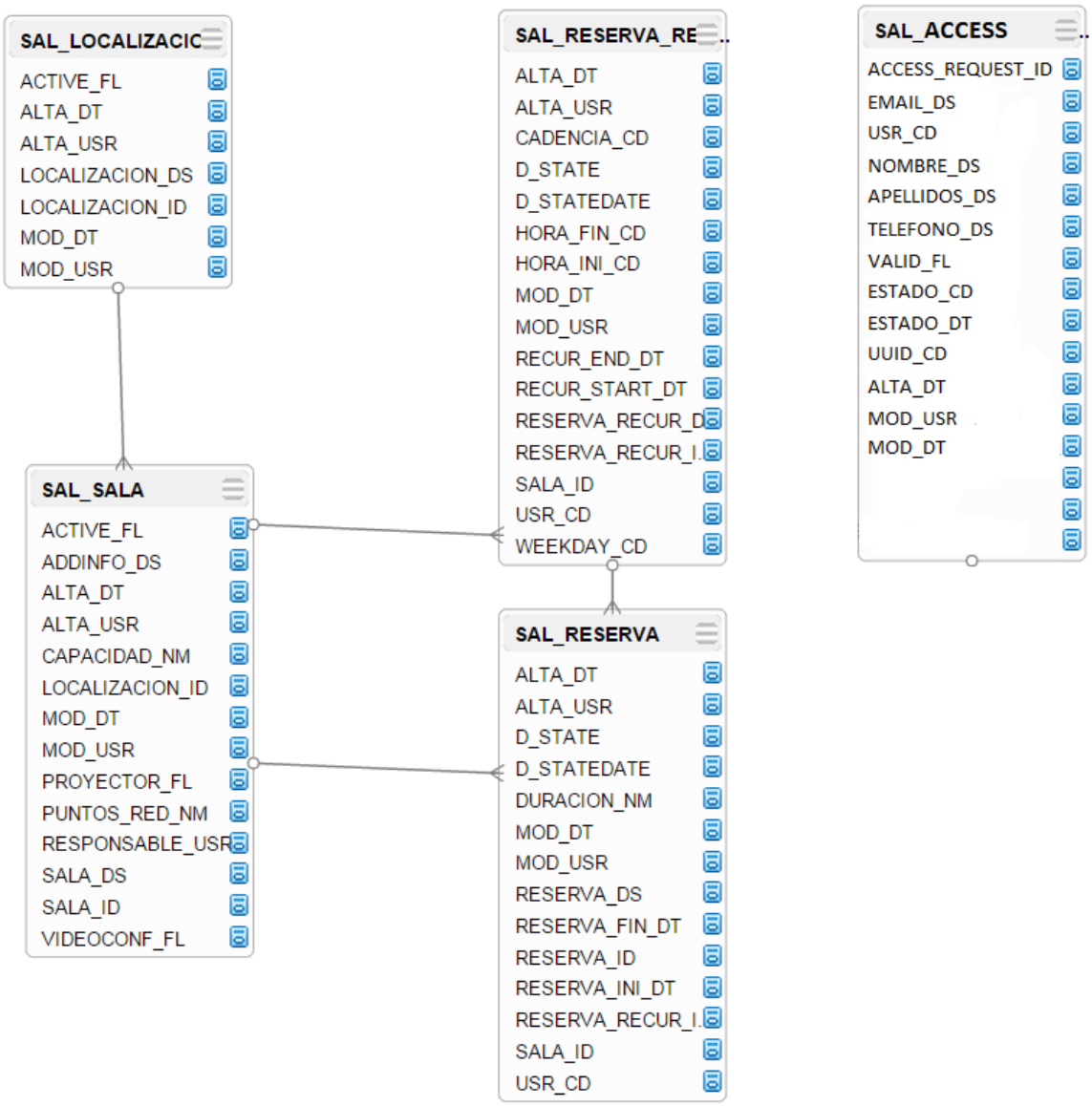


Tabla 3-1 ER BBDD

Se han definido 5 tablas para este sistema:

- **SAL_ACCESS_REQUEST**: Tabla que contiene la información para las solicitudes de acceso al sistema. Tiene una relación (aunque no se muestre en el diagrama) con la tabla de Usuario mediante el campo USR_CD.

Atributos de SAL_ACCESS_REQUEST		
Campo	**Formato**	**Descripción**
APELLIDOS_DS	Varchar(100)	Apellidos del solicitante
D_STATE	Varchar(25)	Estado de la solicitud
D_STATEDATE	Fecha u Hora	Fecha del cambio de estado
EMAIL_DS	Varchar(128)	Email del solicitante
ALTA_DT	Fecha u hora	Fecha de alta
MOD_DT	Fecha u hora	Fecha de modificación
ACCESS_REQUEST_ID	Entero	ID de la tabla
NOMBRE_DS	Varchar(100)	Nombre del solicitante
TELEFONO_DS	Varchar(20)	Teléfono del solicitante
UUID_CD	Varchar(50)	Campo para validar el email
VALID_FL	Bit	Indica si la solicitud ha sido verificada
MOD_USR	Entero	Usuario de modificación
USR_CD	Entero	Usuario de la solicitud

Tabla 3-2 SAL_ACCESS_REQUEST

- SAL_LOCALIZACION: Tabla con toda la información de las localizaciones del sistema. Simplemente se basa en un campo que indica si está activa o no y el nombre de la localización.

Atributos de SAL_LOCALIZACION		
Campo	**Formato**	**Descripción**
ACTIVE_FL	Bit	Indica si la localización está activa
ALTA_DT	Fecha u Hora	Fecha de alta
MOD_DT	Fecha u Hora	Fecha de modificación
LOCALIZACION_ID	Entero	ID de la tabla
LOCALIZAION_DS	Varchar(100)	Nombre de la localización
ALTA_USR	Entero	Usuario de alta
MOD_USR	Entero	Usuario de modificación

Tabla 3-3 SAL_LOCALIZACION

- SAL_SALA: Tabla que contiene todas las salas del sistema y sus características. Las salas pertenecen a una localización, por lo que tienen una relación con la tabla SAL_LOCALIZACION mediante el campo LOCALIZACION_ID. El campo RESPONSABLE_USR es una relación con la tabla USUARIOS, y en ella se guardará el código único del usuario que es responsable de la sala(Campo opcional)

Atributos de SAL_SALA		
Campo	**Formato**	**Descripción**
ACTIVE_FL	Bit	Indica si la sala está activa
CAPACIDAD_NM	Entero	Capacidad de la sala
ALTA_DT	Fecha u hora	Fecha de alta
MOD_DT	Fecha u hora	Fecha de modificación
LOCALIZACION_ID	Entero	ID de la localización a la que pertenece
SALA_ID	Entero	ID de la tabla
SALA_DS	Varchar(100)	Nombre de la sala
PUNTOS_RED_NM	Entero	Puntos de red en la sala
ADDINFO_DS	Varchar(500)	Descripción breve de la sala
PROYECTOR_FL	Bit	Indica si tiene proyector
VIDEOCONF_FL	Bit	Indica si tiene videoconferencia
ALTA_USR	Entero	Usuario de alta
MOD_USR	Entero	Usuario de modificación
RESPONSABLE_USR	Entero	Usuario responsable de la sala

Tabla 3-4 SAL_SALA

- SAL_RESERVA: Tabla con los atributos referentes a las reservas individuales del sistema. En caso de ser una reserva individual generada a partir de una recurrente, el campo RESERVA_RECUR_ID tendrá el valor de la reserva recurrente que la ha generado, por lo que tiene una relación con la tabla SAL_RESERVA_RECUR_ID. Las reservas se realizan a nombre de un usuario, por lo que también existe una relación con la tabla USUARIOS. Todas las reservas tienen que tener una sala, cuya relación se realiza con el campo SALA_ID.

Atributos de SAL_RESERVA		
Campo	**Formato**	**Descripción**
D_STATE	Varchar(25)	Estado de la reserva
D_STATEDATE	Fecha u hora	Fecha del cambio de estado
DURACION_NM	Decimal (10,)	Duración de la reserva
RESERVA_FIN_DT	Fecha u hora	Fecha de fin de la reserva
RESERVA_INI_DT	Fecha u hora	Fecha de inicio de la reserva
ALTA_DT	Fecha u hora	Fecha de alta
MOD_DT	Fecha u hora	Fecha de modificación
RESERVA_ID	Entero	ID de la tabla
RESERVA_RECUR_ID	Entero	ID de la reserva recurrente a la que pertenece
SALA_ID	Entero	ID de la sala que se reserva
RESERVA_DS	Varchar(100)	Nombre de la reserva
ALTA_USR	Entero	Usuario de alta
MOD_USR	Entero	Usuario de modificación
USR_CD	Entero	Usuario al que pertenece la reserva

Tabla 3-5 SAL_ACCESS_REQUEST

- SAL_RESERVA_RECUR: Tabla que guarda la información de las reservas recurrentes del sistema. Mediante el campo SALA_ID se relaciona con la tabla SAL_SALA. Todas las reservas recurrentes tienen un usuario asignado, el cual, como en las anteriores se relación con la tabla USUARIOS mediante USR_CD.

Atributos de SAL_RESERVA_RECUR		
Campo	**Formato**	**Descripción**
D_STATE	Varchar(25)	Estado de la reserva
D_STATEDATE	Fecha u hora	Fecha del cambio de estado
CADENCIA_CD	Entero	Código de la cadencia
RESERVA_RECUR_DS	Varchar(250)	Nombre de la reserva
ALTA_DT	Fecha u hora	Fecha de alta
MOD_DT	Fecha u hora	Fecha de modificación
HORA_FIN_CD	Entero	Hora de finalización
HORA_INI_CD	Entero	Hora de inicio
RESERVA_RECUR_ID	Entero	ID de la tabla
SALA_ID	Entero	ID de la sala que se reserva
RECUR_END_DT	Fecha u hora	Fecha final de la reserva recurrente
RECUR_START_DT	Fecha u hora	Fecha inicial de la reserva recurrente
WEEKDAY_CD	Entero	Código con el día de la semana de la reserva
ALTA_USR	Entero	Usuario de alta
MOD_USR	Entero	Usuario de modificación
USR_CD	Entero	Usuario al que pertenece la reserva

Tabla 3-6 SAL_RESERVA_RECUR

3.4 Conceptos del Sistema

El sistema está compuesto por distintos tipos de clases: **Conceptos**, **Taxonomías** y **Clases Técnicas**

Los conceptos definen los atributos y funciones que tienen los objetos de estos un objeto de ese concepto. En los conceptos también se definen las consultas (querys) que se utilizarán para recuperar objetos de la BBDD

El sistema desarrollado se divide en conceptos. En este apartado se explican los conceptos necesarios para el sistema y se describen brevemente algunas de las características que tendrán, así como sus modelos de estados en caso de tenerlos y algunos de sus atributos.

Las taxonomías son colecciones de objetos que tienen características en común. En este proyecto se han necesitado 4 taxonomías que se describen a continuación.

3.4.1 AccessRequest

Solicitudes de acceso, como hemos indicado en el apartado anterior tiene persistencia en base de datos. Contendrá la información del usuario que ha realizado la solicitud de acceso. Las solicitudes de acceso pueden ser gestionadas solamente por el administrador del sistema, el cual aprobará o rechazará las solicitudes mediante un BPM.
Las solicitudes de acceso pueden estar en los siguientes estados:

- Espera: Estado inicial de la clase. Se queda en este estado hasta que el usuario confirma la dirección de email.
- Pendiente: El usuario ha confirmado su email y se queda a la espera de que un administrador tramite la petición.
- Cancelada: Si un usuario registra varias solicitudes con el mismo email, las anteriores solicitudes pasarán automáticamente a canceladas
- Aprobada: El administrador del sistema ha permitido la solicitud de acceso
- Denegada: El administrador del sistema ha rechazado la solicitud de acceso

3.4.2 Localización

Concepto con persistencia en base de datos que contiene la información de las localizaciones de la aplicación. Esta clase cuenta con página de alta, consulta y modificación. Las localizaciones tienen 2 estados, Activo e Inactivo. Si el estado es activo, se pueden realizar reservas en las salas que pertenezcan a esta localización, por el contrario, si está inactiva, no se podrán realizar más reservas en las salas que pertenezcan a esta localización.

3.4.3 Reserva

Concepto con persistencia en base de datos que contiene la información de las reservas (individuales) realizadas en el sistema. Este concepto cuenta con página de alta, modificación y consulta. Las reservas se realizan sobre las salas y para un usuario determinado.
Las reservas pueden estar en 4 estados:

- Pendiente de aprobación: Cuando una sala tiene un responsable, este tiene que aprobarla, por lo que el estado inicial de la reserva será este.
- Espera Fin: Cuando una reserva ha sido aprobada o no necesita aprobación se mantendrá a la espera de que finalice la reserva.
- Denegada: Estado final. Cuando una sala tiene responsable y este rechaza la solicitud de reserva.
- Finalizada: Estado final. La reserva estaba en espera fin y se ha realizado correctamente.

3.4.4 Reserva recurrente

Concepto con persistencia en base de datos. Una reserva recurrente es aquella que se realiza frecuentemente, por lo que se da la posibilidad a los usuarios de reservar una sala durante un periodo de tiempo en el día indicado. Mediante un trigger, se generarán las reservas individuales de cada reserva recurrente. Al igual que las reservas individuales, las reservas recurrentes se realizan sobre una sala determinada y para un usuario determinado.
Las reservas recurrentes tienen los siguientes estados:

- Pendiente de aprobación: Si la sala indicada tiene responsable, la reserva tiene que ser aprobada.
- Activa: Si la reserva ha sido aprobada o no necesita aprobación, la reserva estará activa hasta que se finalicen todas sus reservas.
- Baja: Si el usuario decide finalizar antes de tiempo la reserva recurrente pasará al estado de baja. También se puede llegar a estado cuando un administrador rechaza tu reserva desde un BPM.
- Finalizada: La reserva recurrente estuvo activa y ha llegado a la fecha final indicada.

3.4.5 Sala

Concepto con persistencia en base de datos. Para definir una sala es necesario asignarle una localización que esté dada de alta en el sistema, un nombre y una serie de datos solicitados en un formulario. Solo un usuario con permisos de administrador puede gestionar las salas del sistema.
Las salas solo se pueden borrar del sistema en caso de que exista ninguna reserva en BBDD asociada a esa sala.
Las salas solamente tienen 2 estados:

- Activa: La sala está disponible y se pueden realizar reservas.
- Inactiva: La sala no está disponible, por lo que no se pueden realizar reservas de ningún tipo sobre ella.

3.4.6 OpenReservaSlot

Clase auxiliar que utilizaremos para poder construir las tablas HTML que se muestran a lo largo del sistema. La función que construye objetos de esta clase se define en el Anexo B.

3.4.7 Globales

Clase técnica que contiene funciones y variables globales que se utilizan en todo el sistema. En este concepto se definen por ejemplo las distintas funciones que contienen los CSS específicos de algunas de las tablas HTML de los buscadores de salas del sistema. Por ejemplo, está definida la función que envía emails a los usuarios y algunas funciones auxiliares que se utilizarán en el sistema.

3.4.1 Webservicetfsl

Clase Técnica en la que se definen todas las funciones que serán integradas en la parte de la aplicación móvil. Las funciones aquí definidas intentan en su mayor parte utilizar funciones propias de cada objeto, por lo que mayormente son funciones que llaman a otras funciones. Algunos ejemplos de las funciones aquí definidas son los siguientes:

```
/*Funcion que borra una reserva**/
public function DelReserva(object objReserva)
{
   var newObj;

   newObj = new ReservaSL(objReserva.RESERVA_ID);
   if(!newObj.IsDelAllowed())
   {
      ThrowErrorSL("Esta reserva no se puede anular");
```

```
      }
      GestorMsg.Reset();
      newObj.ValDeleteGMsg();
      ThrowErrorSLFromGestorMsg();
      newObj.DeleteObject();
   }
/**Obtiene la lista de las reservas del usuario*/
   public function GetListaMisReservas()
   {
      var arrayReserva, objAp;

      objAp = Ap_Reserva.TemplateObj();
      arrayReserva = objAp.GetListaMisReservas();

      return arrayReserva;
   }
```

3.4.2 OpenDia

Concepto auxiliar necesario para saber los días de la semana y en el caso de las reservas recurrentes, será necesario acceder a esta clase para saber si la reserva recurrente se realiza, por ejemplo un lunes o un martes.

3.4.3 OpenHoras

Taxonomía que contiene los valores de todas las horas válidas que pueden tomar las reservas en el sistema. Los valores de esta taxonomía se definen mediante unas variables globales definidas en Globales (HORADAY_INI y HORADAY_FIN). Los valores que toma son desde la HORA_INI hasta HORA_FIN con intervalos de 15 minutos. El código representa los minutos transcurridos a lo largo del día de esa hora. En nuestro sistema, la hora de inicio son las 8AM y la hora de fin las 8PM.

Código	Valor
480	8
495	8:15
510	8:30
525	8:45
…	…
1200	20:00

Tabla 3-7 Horas

3.4.4 OpenDuración

Taxonomía que fija la duración por la que se puede filtrar en las búsquedas de salas que se quiera especificar la duración mínima de la reserva. La lógica que sigue es muy parecida a la de OpenHora, solo que en este caso empieza en 15 minutos y acaba en (HORA_FIN – HORA_INI). Para nuestro sistema toma los siguientes valores.

Código	Valor
15	15m
30	30m
45	45m
60	1h
…	…
720	Día entero

Tabla 3-8 Duración

3.4.5 TipoCadencia

Taxonomía con los posibles valores que puede tomar la cadencia en una reserva recurrente. Los valores de esta taxonomía son los siguientes:

Código	Valor
0	Cada semana
1	Cada 1º semana del mes
2	Cada 2º semana del mes
3	Cada 3º semana del mes
4	Cada última semana del mes

Tabla 3-9 Cadencia

3.4.6 TipoHorario

Taxonomía para realizar búsquedas de sala especificando el horario deseado. Puede tomar los siguientes valores.

Código	Valor
M	Mañana
T	Tarde
N	Indiferente

Tabla 3-10 Horario

3.4.7 Seguridad

La seguridad del sistema se genera mediante SCooP. En el sistema existen 2 roles, Usuario normal y Administrador. Al definir la seguridad, indicamos a que secciones va a tener acceso el usuario del sistema. Este proyecto va a contar con 2 pestañas en la parte superior, “Reserva de salas” que será accesible por todos los usuarios y “Salas (Admin)” pestaña que solo tendrá acceso el administrador y desde la que gestionará las localizaciones, salas y solicitudes de acceso.
La distinción de los roles también servirá para, por ejemplo, al consultar reservas de cualquier persona si el usuario es administrador puede modificar o cancelar esta reserva.

4 Desarrollo

En este apartado se describen los puntos más importantes que se han llevado a cabo al desarrollar el proyecto.

4.1 Localizaciones

El módulo de localizaciones solo es accesible para los administradores. Este módulo cuenta con funciones de alta, consulta y modificación de localizaciones. Para dar de alta una consulta solamente hay que indicar el nombre de la misma. En la consulta de las localizaciones se puede ver una lista que muestra todas las salas que tiene asociada la localización consultada, así como los datos más importantes de las mismas. En la figura 4-1 se puede ver la consulta de una localización.

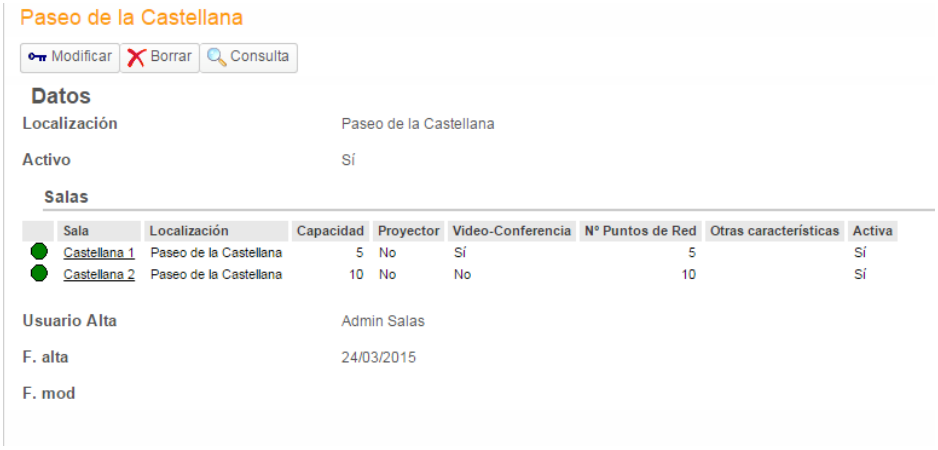


Figura 4-1: Localización

Como se indica en las especificaciones del proyecto, las localizaciones solo pueden ser gestionadas por los administradores, por lo que si un usuario normal consulta una localización (desde las reservas o salas se puede acceder a estas) no puede modificar ni borrar dicha localización. Esto se define con el siguiente código.

```
/**Función Click*/
public function Click()
{
	super.Click();
	if(!GlobalesSL.UserHasPriv("ADMIN_SALA"))
	{
		List_RemoveItem(this,"BETMENUS","ADMINISTRABLE","-");
	}
}
```

4.2 Salas

Las salas solo pueden ser gestionadas por los administradores. Para dar de alta una sala hay que rellenar un formulario como el que se muestra en la figura 4-2.

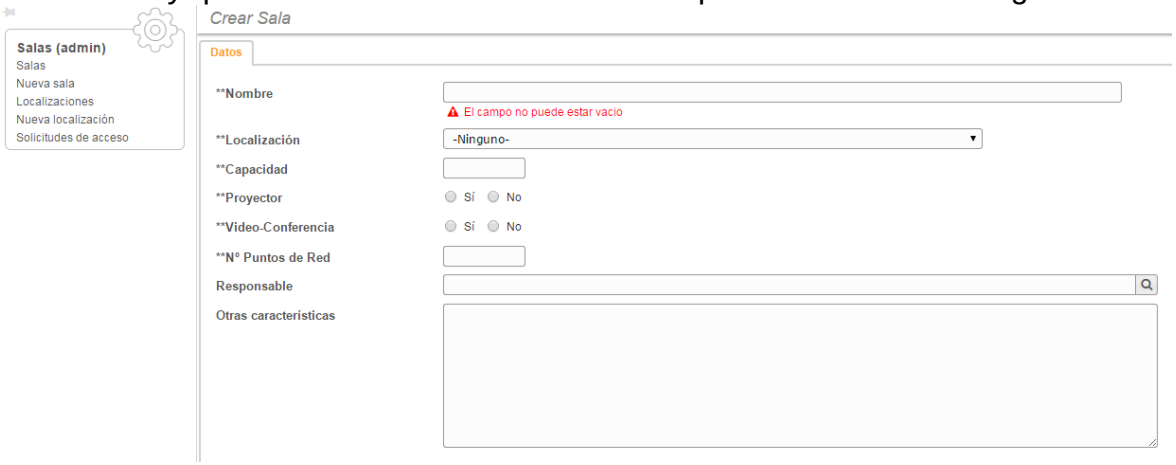


Figura 4-2: Crear sala

Los campos marcados con asteriscos (**) son obligatorios a la hora de dar de alta una sala. Como ya se ha explicado anteriormente el responsable es opcional. En caso de tener responsable, todas las peticiones de reserva que se realicen sobre la sala tendrán que ser aprobadas por dicho responsable mediante un BPM.
En la figura 4-3 se puede ver la consulta de una sala. En esta consulta se pueden ver todas las reservas realizadas sobre la sala, tanto las recurrentes como las individuales. Además, desde la consulta se puede activar/desactivar dicha sala. Solo se puede desactivar si no tiene reservas pendientes.
Las listas de reservas han sido construidas mediante controles en SCooP. Estos controles crean objetos de tipo Reserva y ReservaRecurrente identificándolos por SALA_ID.

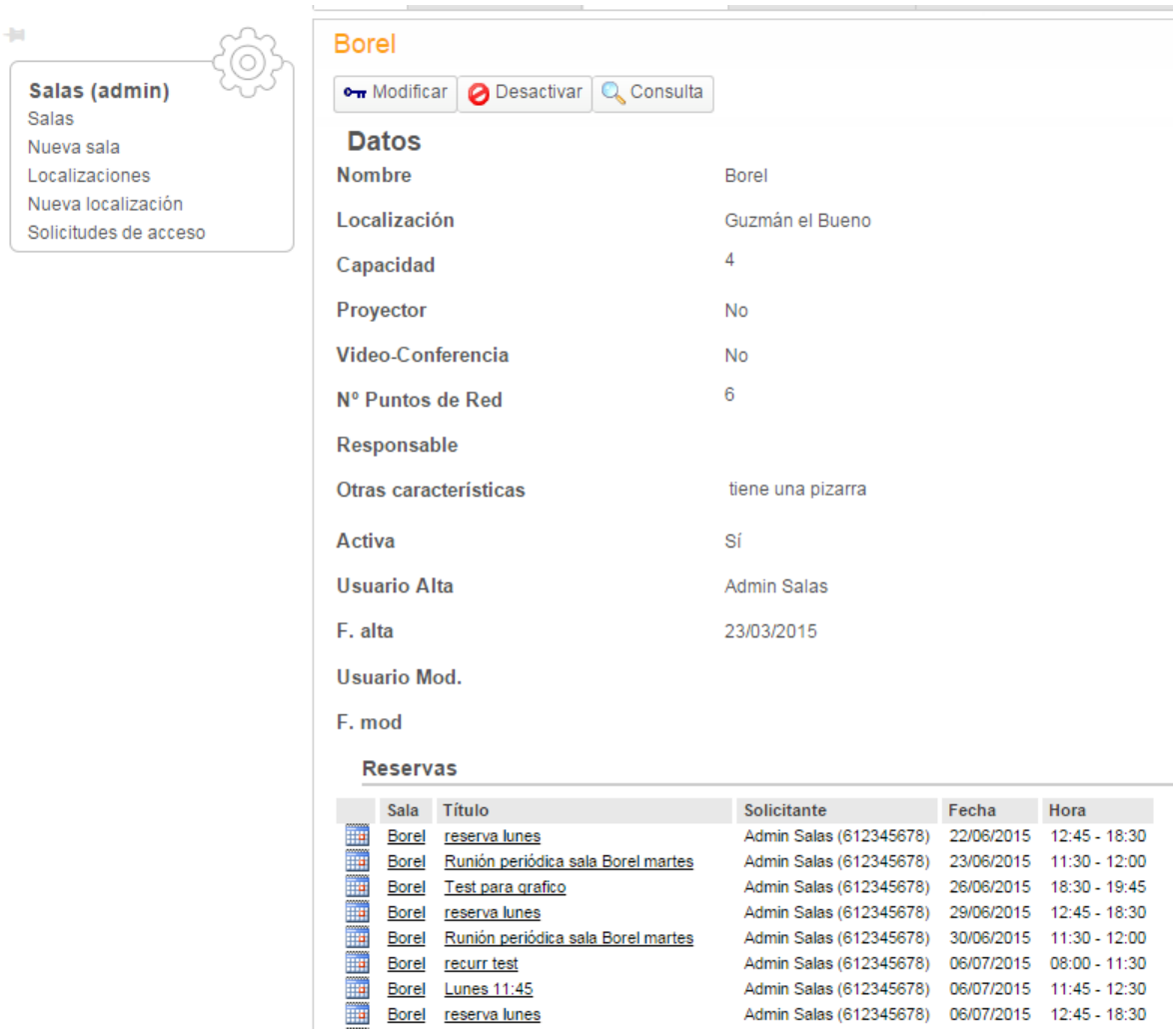


Figura 4-3: Consulta sala

A continuación se muestra el código con el que se comprueba si existen reservas en la sala. En el primer "if" comprobamos si existen objetos de tipo RESERVASL (Reserva individual) que no estén finalizadas en esa sala. La consulta SQL ejecutada es la siguiente,: `"SELECT * FROM SAL_RESERVA WHERE R.SALA_ID = #num,SALA_ID; AND R.ESTADO_CD NOT IN ('FINALIZADA')"` Donde #num,SALA_ID; es la ID de la sala del objeto SALA (objeto de referencia). Si existe alguna reserva, mostramos un mensaje por pantalla y no dejamos marcar como inactiva la sala. El segundo "if" es análogo al primero solo que obteniendo los datos de la tabla SAL_RESERVA_RECUR.

```
/**Función SetInactivo*/
   public function SetInactivo()
   {
      if (ExistsObjects("RESERVASL", "BUILD_NOBAJA_FROM_SALA_ID", this))
      {
         iBetMessage("Atención",   "No se puede desactivar esta sala porque
tiene reservas activas.");
         return false;
      }
      if (ExistsObjects("RESERVARECURSL", "BUILD_NOBAJA_FROM_SALA_ID", this))
      {
         iBetMessage("Atención",   "No se puede desactivar esta sala porque
tiene reservas activas.");
```

```
            return false;
        }
        iBetMessage("Atención",  "La sala será desactivada, ¿desea continuar?",
"BTN_SETINACTIVO_CONTINUE");
        return false;
    }
```

4.3 Solicitudes de acceso

Las solicitudes de acceso se realizan completando un formulario que solicita el nombre, apellido, email y teléfono del solicitante. Para controlar estas solicitudes de acceso se ha desarrollado un buscador que filtra las solicitudes. Los parámetros por los que se puede filtrar en este buscador son los siguientes: Nombre, Apellidos, Email, Teléfono, Fecha de alta de la solicitud (se puede definir un intervalo de fechas) y el estado de las solicitudes a buscar. En la figura 4-4 se puede ver el resultado de una búsqueda sin ningún tipo de filtros. Para facilitar al usuario en qué estado se encuentra cada solicitud se aplican distintos iconos con colores que indican el estado. El buscador se realiza con una query dinámica que tendrá las clausulas especificadas en el buscador.

Resultado de búsqueda de solicitudes de acceso

Ref	Nombre	Email	Teléfono	F.Solicitud	Estado
32	SA	m@m.com	m@m.com	23/04/2015	Denegada
33	123 123	m@m.com	1233	23/04/2015	Espera
10	123 123	123@m.com	123	22/04/2015	Espera
11	1234 1234	1234@m.com	123	22/04/2015	Espera
18	1234 1234	1234@m.com	1234	22/04/2015	Denegada
19	1234 1234	1234@m.com	1234	22/04/2015	Pendiente
20	12345 12345	12345@m.com	12345	22/04/2015	Pendiente
13	2345 2354	123@m.com	1324	22/04/2015	Pendiente
21	aaa aaa	aaa@m.com	aaa	22/04/2015	Pendiente
42	alberto Gimeno	a@g.mui	4152	23/04/2015	Pendiente
27	bbb bbb	bbb@b.com	bbbb	22/04/2015	Aprobada
3	bfzb bfzb	dsfsdf@vs.fs	csdf	21/04/2015	Espera
44	dasdasd asdasd	asdasd@asdasd.com	asdasd	23/04/2015	Denegada
29	den den	den@den.com	den	22/04/2015	Denegada
28	denegada denegada	denegada@m.com	denegada	22/04/2015	Denegada
24	nuevoUser nuevoUser	nuevoUser@user.com	123456789	22/04/2015	Aprobada
67	prueba 1	prueba@prueba.com	123	24/04/2015	Cancelada
68	prueba 2	prueba@prueba.com	123	24/04/2015	Cancelada
69	prueba 3	prueba@prueba.com	123	24/04/2015	Cancelada
70	prueba 4	prueba@prueba.com	123	24/04/2015	Cancelada

1-20 / 57

Figura 4-4: Buscador Peticiones Acceso

4.4 Menú lateral

El menú de acceso a las distintas opciones de los usuarios normales se ha ocultado en una clase llamada AP_RESERVA de modo que desde esta clase se oculta todo el funcionamiento del resto de ficheros de funciones. Podría decirse que la utilizamos a modo de web service. Por ejemplo para obtener la lista de reservas recurrentes o crear una nueva reserva recurrente ocultamos el código de la siguiente forma.

```
/**Función MisResRecur*/
  public function MisResRecur()
```

```
{
   var objAp = AP_ReservaRecur.TemplateObj();
   objAp.Dosfilter("DOSFILTER_DEF");
}
/**Función NewResRecur*/
public function NewResRecur()
{
   var objAp = AP_ReservaRecur.TemplateObj();
   objAp.NewItem();
}
```

Las opciones del menú lateral son las mostradas en la figura 4-5. Mis reservas obtiene la lista de reservas del usuario, Nueva Reserva nos dirige a la página para crear una nueva reserva, Buscar Sala abre el buscador de salas, Mis reservas recurrentes muestra las reservas recurrentes del usuario y Nueva reserva recurrente que nos abre la página de dar de alta una reserva recurrente.

Figura 4-5: Menú lateral

4.5 Buscar Sala

El buscador de salas nos indica mediante una tabla HTML las salas disponibles para los parámetros deseados. El buscador incorpora 2 botones para aumentar y disminuir el día de búsqueda rápidamente. Para este buscador se muestra toda la información de las salas en un día especificado. Si no indicamos ninguna localización en el buscador se buscan salas en todo el sistema ordenando el mostrado de las salas en el HTML por orden alfabético de localizaciones primero y de salas a continuación. En el HTML se muestran todas las reservas individuales tanto aprobadas como pendientes de aprobar. Si una sala no acepta más reservas o no cumple con los requisitos de búsqueda especificados no se mostrará en la tabla. Para realizar una reserva desde esta tabla el usuario debe seleccionar una reserva libre (indicadas en verde) y el sistema, debido a que el HTML incorpora links, navegará a la página de alta de reserva con los datos que se han indicado en el buscador. En la figura 4-6 se puede ver el buscador desarrollado.

La función que crea el HTML del buscador se adjunta y explica en el **Anexo A - GetTableHTML.**

La construcción de las reservas del HTML se realiza mediante la función del **Anexo B - OpenReservaSlot Build.** A esta función se le pasan como objeto de referencia uno de tipo AP_RESERVA y un constructor distinto a BUILD_PDTE_FROM_SESSION_USR para que nos devuelva todas las reservas de la localización indicada sin indicar el usuario.

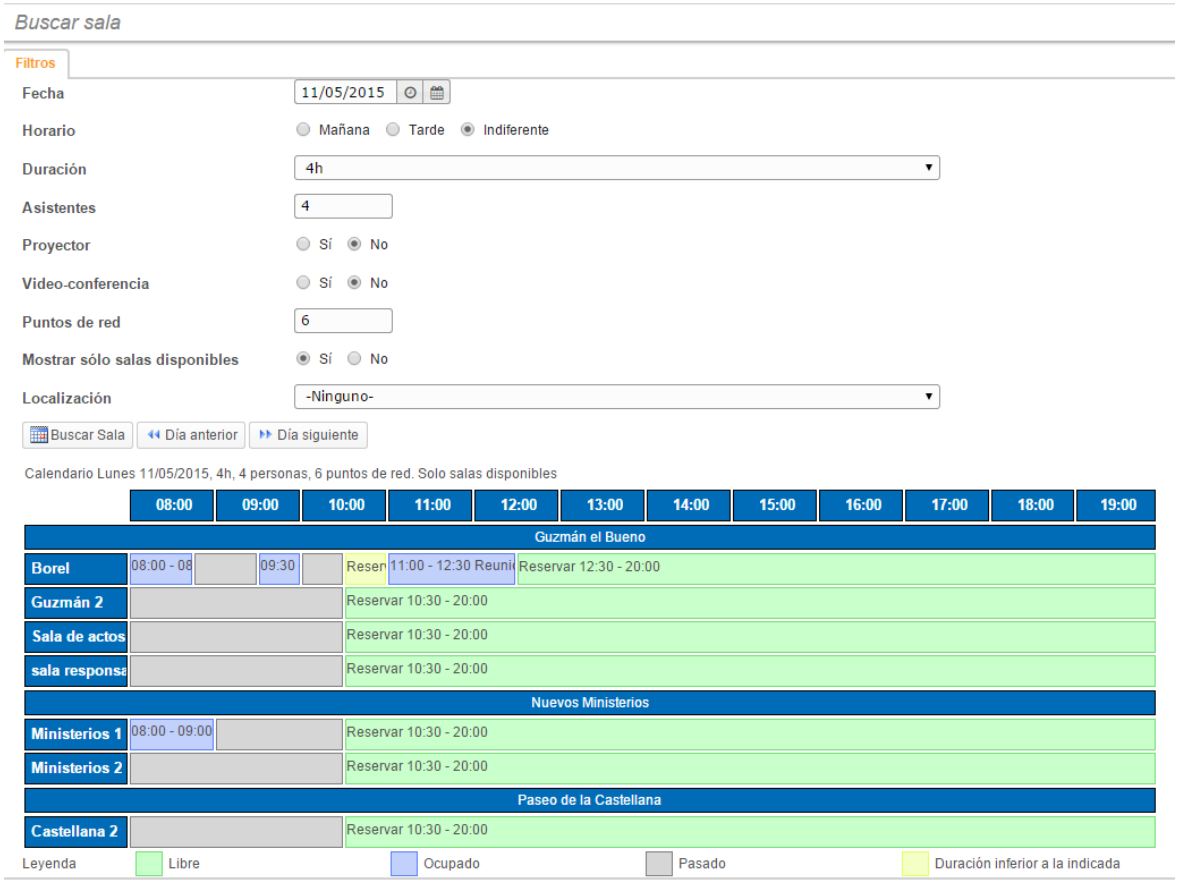


Figura 4-6: Buscar sala

4.6 Reservas

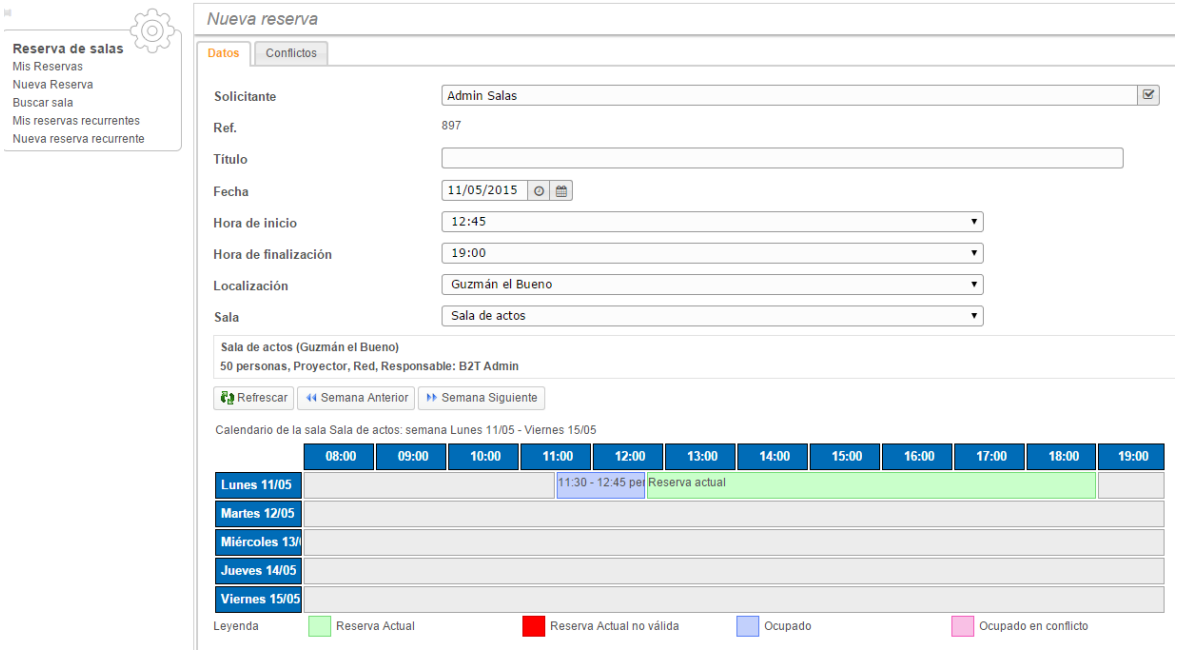


Figura 4-7: Nueva reserva

Para crear una reserva se tienen que introducir una serie de datos obligatorios y cumplir con los requisitos. Para realizar la reserva se ejecuta una función de validación que comprueba que los datos introducidos son correctos y, si existen conflictos con otras reservas se avisa al usuario y se muestra una pestaña con las reservas que crean el conflicto. El código de validación se realiza mediante 2 funciones. ValPgAlta comprueba que los datos introducidos son correctos y en caso de necesitar aprobación avisa al usuario mediante un mensaje por pantalla. Si el usuario confirma que desea hacer la reserva se ejecuta la función OnInsert, la cual introduce en BBDD la reserva y en caso de necesitar aprobación crea un BPM.

```
/**Función ValPgAlta*/
   public function ValPgAlta()
   {
      if(!ValDatos()) return false;
      if(GlobalesSL.IsBpmeRequired(this))
      {
         iBetMessage("Atención", "Su solicitud de reserva tendrá que ser
aprobada por el responsable de la sala." +
         "<br/>¿Desea continuar?","BTN_VALPGALTACONTINUE");
         return false;
      }
      return ValPgAltaContinue();
   }

/**Función OnInsert*/
   public function OnInsert()
   {
      var startBpmFl = false;
      SetHorarioFromFechaHora();
      AsignaDatosAlta();
      if(GlobalesSL.IsBpmeRequired(this))
      {
         startBpmFl = true;
      }
      else
      {
         SetProperty("AprobadaAutoFl",true);
      }
      super.OnInsert();
      if(startBpmFl)
      {
            VariablesEventos.CreateObjEventAdHoc(this,"SAL_STARTBPM_RESERVA"
      );
      }
      SetProperty("AprobadaAutoFl",false);
      SetProperty("ReservaAltaFl",false);
      Reload();
   }
```

El HTML mostrado en la página de solicitud de reserva es muy parecido al descrito en el **Anexo 1 – GetTableHTML()**.
Las reservas se pueden consultar, modificar y cancelar.

Para acceder a la lista de reservas que ha realizado un usuario hay ir a la opción “Mis Reservas” del menú lateral. Se ejecuta la función del Anexo B con el constructor BUILD_PDTE_FROM_SESSION_USR. Al indicar este constructor la función nos devuelve una lista con todas las reservas realizadas por este usuario

y la lista obtenida se muestra en pantalla y así el usuario tiene acceso a todos y puede consultarlas, modificarlas o borrarlas.

Para modificar una reserva las funciones de validación que se realizan en el sistema son las mismas que cuando se da de alta. Existe una excepción y es que cuando una reserva sobre una sala con responsable se quiere modificar, se puede llegar a tener que regenerar la solicitud de reserva y volver a lanzar un BPM. La función que valida si estos cambios son suficientes para relanzar un BPM es la siguiente:

```
/**Función CambiosSignificativos*/
   public function CambiosSignificativos(var objRef)
   {
      var iniDateDiffMin, duracionDiffMin;
      var objSala, reservaRecFl;
      //Si se cambia la sala
      if(objRef.SALA_ID != objRef.GetObjectValueOf("SALA_ID")) return true;
      reservaRecFl = HasHeritage(GetObjClassId(objRef),"RESERVARECURSL");
      if(reservaRecFl)
      {
         // Casos en los que siempre hace falta aprobar
         // Si se cambia el día de la semana
         if(objRef.WEEKDAY_CD    !=    objRef.GetObjectValueOf("WEEKDAY_CD"))
return true;
         // Si se cambia la cadencia menos es caso de Antes Cada semana y
ahora solo alguna semana
         if((objRef.CADENCIA_CD != objRef.GetObjectValueOf("CADENCIA_CD")) &&
            (objRef.GetObjectValueOf("CADENCIA_CD") != 0)) return true;
         // F. Inicio mas temprano
         if(objRef.RECUR_START_DT                                          <
objRef.GetObjectValueOf("RECUR_START_DT")) return true;
         // F. fin más tardío
         if(!empty(objRef.GetObjectValueOf("RECUR_END_DT"))                &&
(empty(objRef.RECUR_END_DT)
          ||                        (objRef.RECUR_END_DT                    >
objRef.GetObjectValueOf("RECUR_END_DT")))) return true;
      }
      else
      {
         if(objRef.RESERVA_DT    !=    objRef.GetObjectValueOf("RESERVA_DT"))
return true;
      }
      if((objRef.HORA_INI_CD  >=  objRef.GetObjectValueOf("HORA_INI_CD"))  &&
(objRef.HORA_FIN_CD <= objRef.GetObjectValueOf("HORA_FIN_CD"))) return false;
      duracionDiffMin   =   (objRef.HORA_FIN_CD   -   objRef.HORA_INI_CD)   -
(objRef.GetObjectValueOf("HORA_FIN_CD")                                     -
objRef.GetObjectValueOf("HORA_INI_CD"));
      iniDateDiffMin            =            objRef.HORA_INI_CD            -
objRef.GetObjectValueOf("HORA_INI_CD");
      return ((duracionDiffMin > 30) || (Math.Abs(iniDateDiffMin) > 60));
   }
```

4.7 Reservas recurrentes

Las reservas recurrentes tienen una implementación muy similar a la de las reservas individuales. Principalmente se diferencian en que la creación de una reserva recurrente tiene que generar las reservas individuales correspondientes desde la fecha indicada hasta dentro de 2 meses desde la fecha actual. Este

procedimiento de generar reservas individuales se realizará mediante un trigger que se ejecuta todos los domingos a las 11 de la mañana.

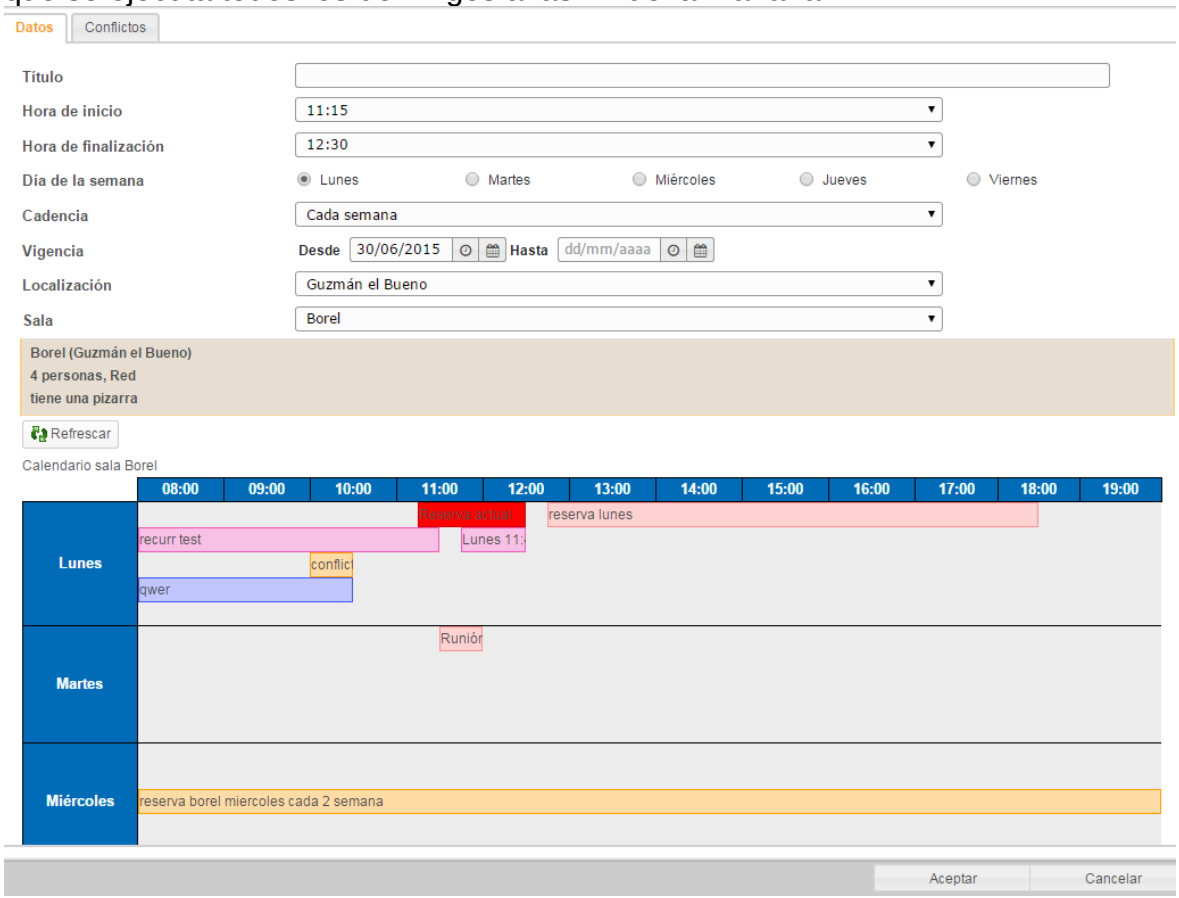


Figura 4-8: Reserva recurrente

En la figura 4-8 se muestra la pantalla de alta de una reserva recurrente. En esta figura se pueden observar los datos necesarios para dar de alta una reserva, la tabla HTML con la información de la sala de lunes a viernes y la reserva que queremos introducir y, en caso de existir, una pestaña que contiene los conflictos que tiene dicha reserva. Este visor, a diferencia del mostrado en las reservas individuales, muestra los conflictos existentes con otras reservas recurrentes y reservas individuales que no están asociadas a otras recurrentes. Este visor se muestra en la figura 4-9. El visor define 2 controles que crean las listas, una con las reservas individuales y otra con las recurrentes.

Figura 4-9: Conflictos recurrente

Para obtener la tabla HTML se ejecuta la función GetReservaTableHTML, explicada en el **Anexo C – GetReservaTableHTML**
Para saber si existe una reserva recurrente en conflicto con la que se quiere meter en el sistema se ejecuta la función ExistsReservaRecurSameTime, esta función obtiene la lista de reservas recurrentes de esa sala en el día especificado entre la hora de inicio y fin concretadas. Si existe alguna reserva activa y que tenga la misma cadencia o su cadencia sea todas las semanas devuelve true y se avisa al usuario.

```
/**Función ExistsReservaRecurSameTime*/
   public function ExistsReservaRecurSameTime()
   {
      var arrayReserva, arrayDaysThis, arrayDaysItem, sameDayCd;
      arrayReserva                                =                                  new
ReservaRecurSL(this,"BUILD_DIFF_FROM_SALA_ID_WEEKDAY_CD_HORA_INI_FIN_CD");
      for(objReserva in arrayReserva)
      {
         if(ExistConflictRecur(objReserva)) return true;
      }
      return false;
   }
   /**Función ExistConflictRecur*/
   public function ExistConflictRecur(var objRecur)
   {
      if(!objRecur) return false;
      if(objRecur.RESERVA_RECUR_ID == RESERVA_RECUR_ID) return false;
      if(objRecur.D_STATE != "ACTIVA") return false;
      if(objRecur.WEEKDAY_CD != WEEKDAY_CD) return false;
      if(objRecur.HORA_INI_CD >= HORA_FIN_CD) return false;
      if(objRecur.HORA_FIN_CD <= HORA_INI_CD) return false;
      if(!empty(RECUR_END_DT)  &&  (objRecur.RECUR_START_DT  >  RECUR_END_DT))
return false;
      if(!empty(objRecur.RECUR_END_DT)     &&     (objRecur.RECUR_END_DT     <
RECUR_START_DT)) return false;
      if((CADENCIA_CD == 0) || (objRecur.CADENCIA_CD == 0)) return true;
      if(CADENCIA_CD == objRecur.CADENCIA_CD) return true;
      return false;
   }
```

Cuando una reserva recurrente se va a registrar en el sistema comprueba si se necesita autorización. Esta comprobación se realiza con la función IsBpmeRequired que simplemente construye la sala de la reserva y devuelve true si la sala tiene responsable.

```
   /**Función IsBpmeRequired*/
   public function IsBpmeRequired(var objRef)
   {
      var iniDateDiffMin, duracionDiffMin;
      var objSala, reservaRecFl;
      if(empty(objRef.SALA_ID)) return false;
      objSala = new SalaSL(objRef.SALA_ID);
      reservaRecFl = HasHeritage(GetObjClassId(objRef),"RESERVARECURSL");
      // Si la reserva es nueva solo depende de la Sala
      if(objRef.GetProperty("ReservaAltaFl"))
      {
         if(!reservaRecFl && !empty(objRef.RESERVA_RECUR_ID)) return false;
      }
      return(!empty(objSala.RESPONSABLE_USR));
```

```
   }
```

La función IsBpmeRequired es llamada desde OnInsert, función que inserta en base de datos la reserva y en caso de necesitar BPM lo crea. Desde esta función también se marca el trigger de creación de reservas individuales para la ejecución automática y por tanto que genere dichas reservas.

```
/**Función OnInsert*/
   public function OnInsert()
   {
      var startBpmFl = false;
      if(empty(RECUR_START_DT)) RECUR_START_DT = ResetTime(SysDate());
      AsignaDatosAlta();
      if(GlobalesSL.IsBpmeRequired(this))
      {
         startBpmFl = true;
      }
      else
      {
         SetProperty("AprobadaAutoFl",true);
      }
      super.OnInsert();
      if(startBpmFl)
      {
         VariablesEventos.CreateObjEventAdHoc(this,"SAL_STARTBPM_RESERVA");
      }
      if(GetProperty("CreateReservaSL"))
      {
         S_Protocol.AddTrigger("Generar                                    reservas
recurrentes",SysDate(),"CREATE_RESERVA_RECUR","TRIGGER_SALAS",RESERVA_RECUR_I
D);
         SetProperty("CreateReservaSL",false);
      }
      SetProperty("AprobadaAutoFl",false);
      SetProperty("ReservaAltaFl",false);
   }
```

Las reservas recurrentes tienen que crear las distintas reservas individuales que tiene cada una, esto se hace mediante un trigger que se ejecuta todos los lunes. En caso de crear una reserva recurrente, se activa el trigger automáticamente y crea todas las reservas individuales para la reserva recurrente creada.
La función que ejecuta el trigger se explica en el apartado 4.9 Triggers.

4.8 Funciones Globales

A continuación se van a describir algunas funciones importantes del sistema y que no se han descrito en los apartados anteriores.
UserHasPriv es la función que indica si el usuario tiene permisos o no para el grupo de seguridad especificado. Se utiliza en los sitios que requieren que solo usuarios con cierto privilegio tengan acceso. En nuestro sistema, un ejemplos sería al modificar salas, solo pueden hacerlo los usuarios que tengan el grupo de seguridad ADMIN.

```
/**Ver si el usuario conectado tiene un grupo de seg*/
   public function UserHasPriv(var grupoSegCd)
   {
      if(empty(grupoSegCd)) return false;
      var ListaMenus = GetSessionDataField("MENULIST");
```

```
   ListaMenus = ListaMenus.ToArray();
   exist(item in ListaMenus)
   {
      item == grupoSegCd;
      return true;
   }
   return false;
}
```

SendEmailSl es la función encargada de enviar emails. Hay que indicarle a quien, el título del email y el texto del mismo. Se utiliza por ejemplo para notificar los errores de generación de reservas individuales mediantes las recurrentes.

```
/**Función SendEmailSL*/
public function SendEmailSL(var emailTo, var subjectDs, var textDs)
{
   if(Empty(emailTo)) return false;
   Mensajeria.M_FROM = Ini_Read("Smtp","From");
   Mensajeria.M_TO = emailTo;
   Mensajeria.M_HTMLMESSAGE = 1;
   Mensajeria.M_SUBJECT = subjectDs;
   Mensajeria.ORIGEN_CD = "SALAS";
   Mensajeria.M_TEXT = textDs;
   Mensajeria.SendEmail();
   return true;
}
```

Hojas de estilo de las tablas: Las hojas de estilo o CSS se encuentran definidas en la clase Globales. Un ejemplo de función con el código CSS de una de las tablas es el siguiente, en concreto es el CSS de la tabla del buscador de salas.

```
/**Función GetStyleSearchSala*/
public function GetStyleSearchSala()
{
   <@
      table.tableSalaCal th
      {
         height: 2em;
         overflow: hidden;
         padding-left: 5px;
      }
      table.tableSalaCal td
      {
         height: 2em;
         overflow: hidden;
      }
      table.tableSalaCal th:hover div.popup
      {
         background: #a3a3a3;
         border-radius: 5px;
         color: #fff;
      }
      table.tableSalaCal td .salaLibre
      {
         height: 100%;
         width: auto;
         vertical-align: middle;
      }
      table.tableSalaCal td .salaLibreInvalid
      {
         height: 100%;
      }
```

```
         table.tableSalaCal td .salaLibreInvalidDuration
         {
            height: 100%;
         }
         table.tableSalaCal td .salaReservada
         {
            height: 100%;
            width: auto;
         }
         table.tableSalaCal td:hover div.popup
         {
            background: #a3a3a3;
            border-radius: 5px;
            color: #fff;
         }
         a.salaNoUnderline
         {
            display:block;
            height: 100%;
             text-decoration: none !important;
         }
         .salaNoUnderline .reservaSlot{
            vertical-align: middle;
         }
         table.tableSalaCal .localizacion{
            overflow: hidden;
            border: 1px solid black;
            text-align: center;
            color: white;
            background: #006CB7;
         }

      @>
      return state;
   }
```

4.9 Triggers

Los triggers tienen un papel muy importante ya que nos permiten seguir navegando por el sistema sin necesidad de esperar a que se ejecuten ciertos procesos. Se han definido 3 triggers, CheckReservaFianlizada, CreateReservaRecur y DeleteReservaRecur.

CheckReservaFinalizada es un trigger que se ejecuta cada 15 minutos y lo que hace es obtener todas las reservas cuya fecha de finalización haya pasado y actualizarlas. Al actualizar estas reservas el estado de las mismas cambiará a Finalizada ya que quiere decir que la reserva se ha finalizado.
La función de definición del trigger es la siguiente, el 900 representa los segundos que tienen que pasar para que se ejecute el trigger de nuevo.

```
S_Protocol.AddLoopTrigger("Finalizar
reservas",900,"CHECK_RESERVA_FIN","TRIGGER_SALAS");
public interface procedure CHECK_RESERVA_FIN("","",CheckReservaFinalizada)
/**CheckReservaFinalizada*/
   public function CheckReservaFinalizada()
   {
      var arrayReserva;
      arrayReserva = new ReservaSL(this,"BUILD_PENDFIN_FROM_GETDATE");
      for(objReserva in arrayReserva)
```

```
      {
         objReserva.OnUpdate();
      }
   }
```

CreateReservaRecur es el trigger encargado de crear todas las reservas individuales de las reservas recurrentes. Se ejecuta todos los domingos a las 11 de la mañana por petición expresa del cliente.

```
S_Protocol.AddLoopTrigger("CreateReservaRecur","W,11:00,D","CREATE_RESERVA_RE
CUR","TRIGGER_SALAS");
public interface procedure CREATE_RESERVA_RECUR("","",CreateReservaRecur)
   /**Función CreateReservaRecur*/
   public function CreateReservaRecur()
   {
      var objResRec, arrayResRec, errorMsg;
      var objUsr, subjDs, textDs;
      var arrayResRecPend, objResRecPend;
      arrayResRec = new Array();
      if(!empty(ADDITIONAL_INFO))
      {
         objResRec = new ReservaRecurSL(ADDITIONAL_INFO);
         arrayResRec.Add(objResRec);
      }
      else
      {
         arrayResRec = new ReservaRecurSL(this,"BUILD_ACTIVA");
      }
      for(objResRec in arrayResRec)
      {
         errorMsg = objResRec.CreateReservaItem();
         if(!empty(errorMsg))
         {
            objUsr = new AdUser(objResRec.USR_CD);
            if(!empty(objUsr.EMAIL_DS))
            {
               subjDs      =      "Error      al      generar      reservas
'%s'".Format(objResRec.RESERVA_RECUR_DS);
               textDs = "Han ocurrido errores al generar reservas '%s' para
las siguientes fechas:%s".Format(objResRec.RESERVA_RECUR_DS,errorMsg);
               GlobalesSL.SendEmailSL(objUsr.EMAIL_DS,subjDs,textDs);
            }
         }
      }
      arrayResRecPend                              =                              new
ReservaRecurSL(this,"BUILD_PEND_APROB_FROM_GETDATE");
      for(objResRecPend in arrayResRecPend)
      {
         objResRecPend.OnUpdate();
      }
   }
```

DeleteReservaRecur es el trigger que se ejecuta cuando un usuario da de baja una reserva recurrente. Este trigger borra las reservas individuales que se habían generado hasta el momento para esa reserva recurrente.

```
public interface procedure DELETE_RESERVA_RECUR("","",DeleteReservaRecur)
   /**Función DeleteReservaRecur*/
   public function DeleteReservaRecur()
   {
      var objResRec;
      if(empty(ADDITIONAL_INFO))
```

```
      {
         system.out.format("Error en DeleteReservaRecur: falta Id de
reserva");
         return;
      }
      objResRec = new ReservaRecurSL(ADDITIONAL_INFO);
      objResRec.DeleteReservaRecur();
   }
```

4.10 WebService

La clase WebService cuenta con toda la funcionalidad que se aplica a la aplicación móvil. En esta clase están definidas todas las funciones que se pueden utilizar en los elementos de tipo WebService de la parte de SMAPPS.
Este WebService no tiene objetos propios, simplemente permite la comunicación de los objetos definidos para la aplicación web con los que se definan en la aplicación móvil. A continuación se muestran algunos ejemplos de funciones.
ModReserva es la función que dado un objeto reserva que proviene del móvil, valida que este objeto sea válido y lo recupera de base de datos y lo modifica con los nuevos datos. Al igual que las funciones de la web, si se necesita lanzar un BPM también lo contempla y lo lanza.

```
   /**Añade una reserva*/
   public function ModReserva(object objReserva)
   {
      var newObj;

      newObj = new ReservaSL(objReserva.RESERVA_ID);
      if(!newObj.IsModAllowed())
      {
         ThrowErrorSL("Esta reserva no se puede modificar");
      }
      newObj.SetProperty("ReservaModFl",true);
      newObj.SetDataFromWs(objReserva);
      GestorMsg.Reset();
      newObj.ValDatosGMsg();
      ThrowErrorSLFromGestorMsg();
      return              (!(GlobalesSL.IsBpmeRequired(newObj)              &&
GlobalesSL.CambiosSignificativos(newObj)));
   }
```

GetListaReservas es la función que obtiene las reservas del usuario que esté en la aplicación.

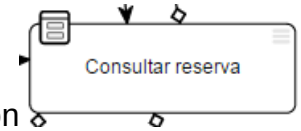
```
   /**Obtiene la lista de las reservas del usuario*/
   public function GetListaMisReservas()
   {
      var arrayReserva, objAp;

      objAp = Ap_Reserva.TemplateObj();
      arrayReserva = objAp.GetListaMisReservas();

      return arrayReserva;
   }
```

4.11 Desarrollo aplicación móvil

La aplicación móvil se ha desarrollado completamente en SCooP. Para el desarrollo de la aplicación se ha definido un esquema que indica el flujo que va a seguir la aplicación acorde a unas reglas. El esquema realizado es el que se muestra en el Anexo C – Esquema SMAPPS
En este esquema se pueden observar distintos tipos de elementos. Las flechas indican el flujo que va a seguir la aplicación. Existen 2 tipos de flechas, las que tienen un cuadrado en el inicio indican que son flujos condicionados por acciones del usuario y las que no tienen cuadrado que son flujos predeterminados
Elementos del esquema:

- Inicio: Marca donde comienza la ejecución del servicio 

- Web Service: Peticiones que se realizan a las funciones de la aplicación definidas en el apartado 4.X 

- Página: Visores que va a contener la aplicación 

- Script: Scripts en JavaScript que se van a ejecutar 

- Gateway: Tomas de decisión de la aplicación 
- Mensaje: Mensaje que se mostrará por pantalla 

La aplicación tiene una base de datos local (conocida como pizarra) capaz de guardar objetos, estructuras, arrays y variables locales. En la figura 4-10 se pueden ver todas las variables definidas para la aplicación, así como las estructuras que utilizan algunos objetos.

Nombre	Tipo	Tipo elemento	Estructura	Descripción corta
arrayDuracion	Lista	-	STC_DURACION	arrayDuracion
arrayHorario	Lista	-	STC_TIPOLOGIA_STRING	arrayHorario
arrayHorasFin	Lista	-	STC_HORAS	Array con las horas finales
arrayHorasIni	Lista	-	STC_HORAS	Array con las horas de inicio disponibles
arrayLocalizaciones	Lista	-	STC_LOCALIZACION	Array con las localizaciones de las salas
arrayReservas	Lista	-	STC_RESERVA	arrayReservas
arraySalas	Lista	-	STC_SALA_SHORT	Array con todas las salas
arraySiNo	Lista	-	STC_TIPOLOGIA_INT	arraySiNo
arrayTipoBusqueda	Lista	-	STC_TIPOLOGIA_STRING	arrayTipoBusqueda
diaActualDt	Elemento	Fecha	-	diaActualDt
horaFinDs	Elemento	String	-	horaFinDs
horaIniDs	Elemento	String	-	horaIniDs
localizacionDs	Elemento	String	-	localizacionDs
objBuscador	Instancia	-	STC_BUSCADOR	objBuscador
objResBusqueda	Instancia	-	STC_SEARCH_RESULT	objResBusqueda
objReserva	Instancia	-	STC_RESERVA	objReserva
objSalaLong	Instancia	-	STC_SALA_LONG	objSalaLong
objTipoBusqueda	Instancia	-	STC_TIPOLOGIA_STRING	objTipoBusqueda
respuestaWS	Elemento	Booleano	-	respuestaWS
STC_BUSCADOR	Estructura	-	-	STC_BUSCADOR
STC_DURACION	Estructura	-	-	STC_DURACION
STC_HORAS	Estructura	-	-	Estructura para guardar los datos de los XMl con las horas
STC_LOCALIZACION	Estructura	-	-	STC_LOCALIZACION
STC_RESERVA	Estructura	-	-	STC_RESERVA
STC_SALA_LONG	Estructura	-	-	STC_SALA_LONG
STC_SALA_SHORT	Estructura	-	-	STC_SALA_SHORT
STC_SEARCH_RESULT	Estructura	-	-	STC_SEARCH_RESULT
STC_TIPOLOGIA_INT	Estructura	-	-	Estructura utilizada para todos los datos con un codigo Ds y otro Cd(entero)
STC_TIPOLOGIA_STRING	Estructura	-	-	Estructura utilizada para todos los datos con un codigo Ds y otro Cd(cadena)

Figura 4-10: Pizarra SMAPPS

Estas variables se utilizan en todos los elementos enumerados anteriormente.

En total la aplicación cuenta con 9 páginas, 6 scripts, 6 mensajes, 5 gateways y 12 funciones de WebService.
Una vez tenemos definido el esquema y las variables de la aplicación tenemos que definir todos los elementos nombrados anteriormente, así como tenemos que dotar de la correcta navegación al sistema.
Las páginas se definen mediante controles, los cuales están asociados a elementos de la pizarra. Por ejemplo un RadioButton está asociado con un array de valores y un objeto que contendrá la selección de dicho array. Las páginas tienen la opción de tener una botonera inferior (footer), cuyos botones se asocian con distintas acciones (flechas con cuadrado en el diagrama). El resultado de definir una página es el siguiente:

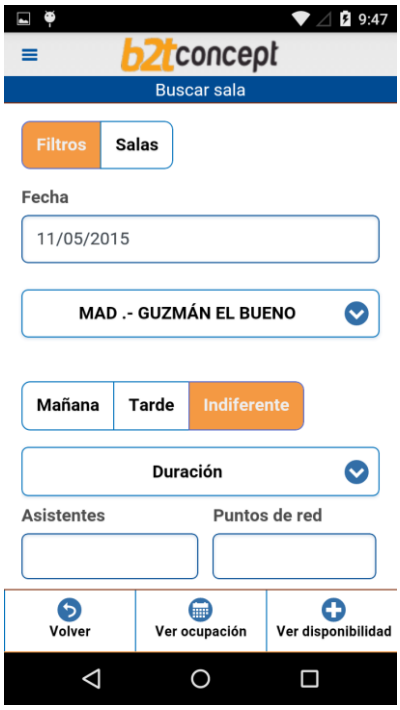


Figura 4-11: Buscar sala APP

Los scripts se definen en JavaScript. En esta aplicación los utilizamos para aumentar o disminuir un día en la búsqueda de salas o para limpiar campos de los objetos. El script mostrado a continuación aumenta un día la fecha del buscador.

```
var fActual = Board.get('objBuscador.RESERVA_DT');
var dia = fActual.getDate();
var mes = fActual.getMonth();
var anio = fActual.getFullYear();
var fechaNueva = new Date(anio ,mes ,dia+1,0,0,0,0);
console.log(fechaNueva);
Board.set('objBuscador.RESERVA_DT', fechaNueva);
```

Los mensajes tienen asociados botones y en ellos se pueden mostrar variables de la pizarra. En este caso es un mensaje con 2 botones (cada uno tiene una acción distinta). El texto del mensaje muestra un texto estándar más el nº de la reserva del objeto objReserva de la pizarra. El código del mensaje sería el siguiente:

```
La sala ha sido reservada con éxito. Su n° de reserva es
{{objReserva.RESERVA_ID}}.
```

Figura 4-12: Mensaje SMAPPS

Los gateways toman decisiones en el sistema en función de los valores que toman los objetos o variables que se le indican. En esta aplicación por ejemplo según el tipo de búsqueda que se indique en el radio de búsqueda de salas se ejecuta un flujo u otro. Dependiendo del estado de la reserva que te devuelva un WebService, muestra o no un mensaje avisando de que la reserva debe ser aprobada o ha sido realizada directamente.

Los elementos WebServices son llamadas a las funciones definidas en la clase WebService del proyecto. Para realizar estas llamadas se le indican la estructura que tienen los elementos del WebServices y los objetos o variables de entrada y salida de los mismos.

Las 9 páginas con las que cuenta la aplicación son las siguientes:

Código Página	Descripción
PG_BUSCAR_SALA	Página con el buscador de salas
PG_CONSULTAR_RESERVA	Página de consulta de reserva
PG_CONSULTARRESERVA2	Página de consulta de reserva
PG_EDIT_RESERVA	Página para editar reservas
PG_INICIAL	Página inicial de la app
PG_MISRESERVAS	Página con las reservas del usuario
PG_RESERVA	Página para realizar una reserva
PG_VER_OCUPACION	Página con los resultados del buscador
PG_VER_SALA	Página con los resultados del buscador

Tabla 4-11 Páginas SMAPPS

El total de funciones que tiene la clase WebService son 10 sin embargo en la aplicación se definen 12 elementos de tipo WebService. Esto es debido a que algunos de elementos realizan la misma llamada de webservice ya que debido a limitaciones a la hora de definir el flujo impedían unificar estos WebService con la misma funcionalidad. En la figura 4-13 se pueden ver a la izquierda los elementos WS de SMAPPS con la función que utilizan de la clase WebService.

Elementos WS

Elemento	Función	Tipo
GetListaSalas	GetListaSalas	Webservice
GetListaLocalizaciones	GetListaLocalizaciones	Webservice
GetListaMisReservas	GetListaMisReservas	Webservice
AddReserva	AddReserva	Webservice
GetReservas	GetListaReservas	Webservice
GetDatosSala	GetInfoSala	Webservice
DelReserva	DelReserva	Webservice
EditReserva	ModReserva	Webservice
GetInfoSala	GetInfoSala	Webservice
GetInfoSala2	GetInfoSala	Webservice
AddReservaContinue	AddReservaContinue	Webservice
ModReservaContinue	ModReservaContinue	Webservice

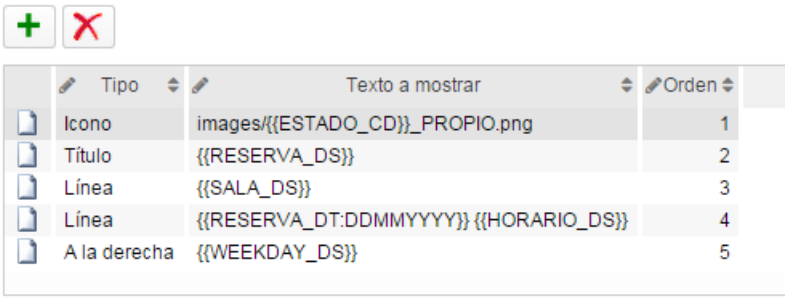
Figura 4-13: WebServices SMAPPS

Los arrays que contienen los tipos de hora, los horarios y las horas de las reservas no se calculan por WS ya que se considera que hacer peticiones a WS para obtener estos datos aumenta el consumo de datos notablemente, sobre todo si son muchos objetos y también mucho más lenta la carga inicial de la aplicación. En este caso los valores se obtienene de ficheros XML. Por ejemplo las Horas de inicio se calculan con el siguiente XML.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
 <HorasSala>
  <HoraSala HoraSalaId="480" HoraSalaDs="08:00" />
  <HoraSala HoraSalaId="495" HoraSalaDs="08:15" />
  <HoraSala HoraSalaId="510" HoraSalaDs="08:30" />
  <HoraSala HoraSalaId="525" HoraSalaDs="08:45" />
  <HoraSala HoraSalaId="540" HoraSalaDs="09:00" />
  <HoraSala HoraSalaId="555" HoraSalaDs="09:15" />
  <HoraSala HoraSalaId="570" HoraSalaDs="09:30" />
  <HoraSala HoraSalaId="585" HoraSalaDs="09:45" />
  <HoraSala HoraSalaId="600" HoraSalaDs="10:00" />
  <HoraSala HoraSalaId="615" HoraSalaDs="10:15" />
  <HoraSala HoraSalaId="630" HoraSalaDs="10:30" />
  <HoraSala HoraSalaId="645" HoraSalaDs="10:45" />
  <HoraSala HoraSalaId="660" HoraSalaDs="11:00" />
  <HoraSala HoraSalaId="675" HoraSalaDs="11:15" />
  <HoraSala HoraSalaId="690" HoraSalaDs="11:30" />
  <HoraSala HoraSalaId="705" HoraSalaDs="11:45" />
  <HoraSala HoraSalaId="720" HoraSalaDs="12:00" />
  <HoraSala HoraSalaId="735" HoraSalaDs="12:15" />
  <HoraSala HoraSalaId="750" HoraSalaDs="12:30" />
  <HoraSala HoraSalaId="765" HoraSalaDs="12:45" />
  <HoraSala HoraSalaId="780" HoraSalaDs="13:00" />
```

```
<HoraSala HoraSalaId="795" HoraSalaDs="13:15" />
<HoraSala HoraSalaId="810" HoraSalaDs="13:30" />
<HoraSala HoraSalaId="825" HoraSalaDs="13:45" />
<HoraSala HoraSalaId="840" HoraSalaDs="14:00" />
<HoraSala HoraSalaId="855" HoraSalaDs="14:15" />
<HoraSala HoraSalaId="870" HoraSalaDs="14:30" />
<HoraSala HoraSalaId="885" HoraSalaDs="14:45" />
<HoraSala HoraSalaId="900" HoraSalaDs="15:00" />
<HoraSala HoraSalaId="915" HoraSalaDs="15:15" />
<HoraSala HoraSalaId="930" HoraSalaDs="15:30" />
<HoraSala HoraSalaId="945" HoraSalaDs="15:45" />
<HoraSala HoraSalaId="960" HoraSalaDs="16:00" />
<HoraSala HoraSalaId="975" HoraSalaDs="16:15" />
<HoraSala HoraSalaId="990" HoraSalaDs="16:30" />
<HoraSala HoraSalaId="1005" HoraSalaDs="16:45" />
<HoraSala HoraSalaId="1020" HoraSalaDs="17:00" />
<HoraSala HoraSalaId="1035" HoraSalaDs="17:15" />
<HoraSala HoraSalaId="1050" HoraSalaDs="17:30" />
<HoraSala HoraSalaId="1065" HoraSalaDs="17:45" />
<HoraSala HoraSalaId="1080" HoraSalaDs="18:00" />
<HoraSala HoraSalaId="1095" HoraSalaDs="18:15" />
<HoraSala HoraSalaId="1110" HoraSalaDs="18:30" />
<HoraSala HoraSalaId="1125" HoraSalaDs="18:45" />
<HoraSala HoraSalaId="1140" HoraSalaDs="19:00" />
<HoraSala HoraSalaId="1155" HoraSalaDs="19:15" />
<HoraSala HoraSalaId="1170" HoraSalaDs="19:30" />
<HoraSala HoraSalaId="1185" HoraSalaDs="19:45" />
</HorasSala>
```

La definición de los controles de tipo lista se realiza como se indica en la figura 4-14, esto quiere decir que de cada objeto que contiene la lista le asigna al Título el atributo RESERVA_DS, estos objetos en la lista tendrán 2 líneas adicionales, en una se muestra el nombre de la sala y en otra la fecha y hora de la reserva en cuestión.



	Tipo	Texto a mostrar	Orden
	Icono	images/{{ESTADO_CD}}_PROPIO.png	1
	Título	{{RESERVA_DS}}	2
	Línea	{{SALA_DS}}	3
	Línea	{{RESERVA_DT:DDMMYYYY}} {{HORARIO_DS}}	4
	A la derecha	{{WEEKDAY_DS}}	5

Figura 4-134: Ctrl Lista SMAPPS

Este control en concreto es el de la lista de mis reservas que se obtienen mediante WebSerice. A los controles se les puede asociar navegación. En este caso si se selecciona un elemento de la lista (son reservas) se hará una petición a WS que nos devuelva todos los datos de la reserva en cuestión. En la figura 4-15 se puede ver cómo queda la página implementada.

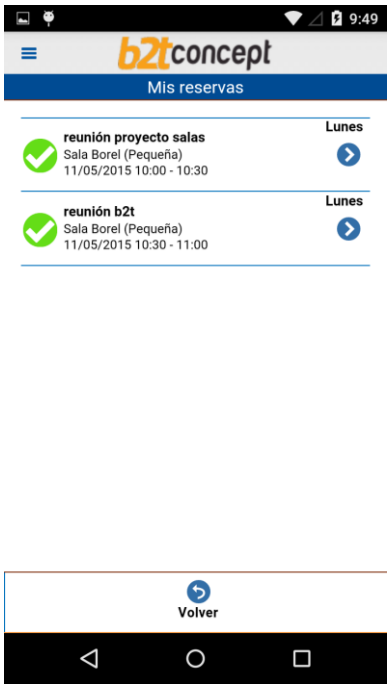


Figura 4-145: Pg_MisReservas SMAPPS

5 Integración, pruebas y resultados

Las pruebas realizadas al sistema han sido principalmente durante el desarrollo, es decir, comprobando que la funcionalidad solicitada en los requisitos era la que realizaba el sistema.

Para realizar estas pruebas nos ayudamos del DEVR ya que cuenta con un depurador y podemos ver por pantalla todo el flujo que se ha ido realizando entre función y función y las ejecuciones de las consultas que se han realizado.
En caso de que hubiera sucedido algo extraño, el DEVR nos avisa mostrando una traza con el error detectado.

En la versión móvil para realizar las pruebas se ha utilizado el inspector de elementos de Google Chrome y mediante la sentencia Console.Log("texto") en los scripts hemos podido seguir el flujo que se iba ejecutando.

Como se comentó en la fase de Diseño, se introdujeron nuevos requisitos en mitad del desarrollo. Esto fue porque se entregó parte del software que se había desarrollado hasta el momento y el cliente, que es el usuario final, realizó distintas pruebas sobre la aplicación web y móvil.

Los resultados de las pruebas han sido satisfactorios ya que hasta la fecha de elaboración de este documento no se han recibido incidencias referentes a la aplicación, que ya está en uso por parte del cliente.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

En este TFG se han desarrollado dos aplicaciones (web y móvil) completamente compatibles entre ellas que permiten la gestión de salas y reservas que se introducen en el sistema.

Los resultados obtenidos están a la altura de lo que esperaba el cliente, ya que desde que se entregó la aplicación en su última versión, no se han recibido incidencias ni peticiones de mejora.

A pesar de todo esto algunos apartados de la aplicación podrían ser mejorados, como por ejemplo la representación de los HTML de las reservas contenidas en las salas.

6.2 Trabajo futuro

El trabajo futuro a realizar sobre la aplicación, debido a que es una aplicación desarrollada para un cliente, será principalmente el de mantenimiento y soporte de la aplicación, es decir si el cliente detecta algún fallo que no se ha detectado en las pruebas o desean realizar cambios. Estos cambios habría que estudiarlos y ver si son posibles llevarlos a cabo.

Cabe destacar que debido a que los BPME son una implementación reciente de la arquitectura B2T, no se ha podido implementar un panel de administración de estos, por lo que se deja como trabajo pendiente y, en cuanto sea posible su implementación se realizará dicho panel de administración.

Referencias

Página web de B2TConcept - www.b2tconcept.com

Repositorio de presentaciones sobre MDE - http://modeling-languages.com/useful-presentations-model-driven-engineering-dsls-uml-eclipse-modeling-technologies-2/#global

MDE en Wikipedia - https://en.wikipedia.org/wiki/Model-driven_engineering

Blog TheEnterPriseArchitect - http://www.theenterprisearchitect.eu/blog/2009/01/15/mde-model-driven-engineering-reference-guide/

Microsoft - https://msdn.microsoft.com/en-us/library/aa964145.aspx?f=255&MSPPError=-2147217396

Glosario

BPM: Proceso de negocio que tiene asociados distintos eventos.

MDE: Model Driven Engineering o Ingenieria Dirigida por Modelos.

Anexos

A Función GetSalaTableHTML

Función que devuelve un HTML para representarlo en distintas páginas de la aplicación. Es de las funciones más complejas que se ha desarrollado ya que mezcla código JCOR con HTML. En la variable state se va guardando todo el código que está entre <@ y @> sin embargo, si en mitad de estas 2 etiquetas queremos evitar que nos meta código, tenemos que ocultarlo con <% Codigo JCOR %> por lo que podemos hacer tablas de arrays recorriéndolos sin que nos muestre el código JCOR escrito.

```
/**Obtener el HTML de ocupacion de Salas en 1 día
   * arrayOpenRes : array OpenReservaSala con las reservas completas del día
ordenadas por Hora Inicio - Libres y ocupadas*/
   public function GetSalaTableHTML(var nuevaReserva = false)
   {
      var ansHtml, arrayHora, colWidth, headerDs, divTitleStr, resWidth;
      var mapSala, arraySalaId, arraySalaOrder, arrayResSala, objSala;
      var arrayOpReserva = new OpenReservaSlot(this,"BUILD_SEARCH");
      if(arrayOpReserva.IsEmpty()) return "";
      arrayHora = new OpenHoraSL(this,"BUILD_ROUND_FROM_HORARIO_CD");
      colWidth = 90.0/(arrayHora.GetSize() * 4);
      mapSala = new strMap();
      arraySalaOrder = new Array();
      for(objOpRes in arrayOpReserva)
      {
         if(!mapSala[objOpRes.SALA_ID])
         {
            objSala = new SalaSL(objOpRes.SALA_ID);
            mapSala[objOpRes.SALA_ID] =
            {
               "SALA_DS"   : objSala.SALA_DS,
               "DESCRI"                                                    :
"<b>%s<br/>%s</b>".Format(objSala.SALA_DS,objSala.GetSalaInfo())
            };
            mapSala[objOpRes.SALA_ID]["RESERVAS"] = new Array();

arraySalaOrder.Add([objOpRes.SALA_ID,objOpRes.SALA_DS,objOpRes.LOCALIZACION_D
S]);
         }
         arrayResSala = mapSala[objOpRes.SALA_ID]["RESERVAS"];
         arrayResSala.Add(objOpRes);
         mapSala[objOpRes.SALA_ID]["RESERVAS"] = arrayResSala;
      }
      arraySalaOrder.Sort(function(x,y)
                          {
                             if(Upper(x[2]) < Upper(y[2])) return -1;
                             if(Upper(x[2]) > Upper(y[2])) return 1;
                             if(Upper(x[1]) < Upper(y[1])) return -1;
                             if(Upper(x[1]) > Upper(y[1])) return 1;
                             return 0;
                          });

      SetTitleSearchDs(true);
      <@
      <table class="tableSalas tableSalaCal">
         <tr>
            <td  width="10%"  style="overflow:  visible;"><%=  TITLE_SEARCH_DS
%></td>
```

```
      <%
      for (var n=0; n<4*arrayHora.GetSize(); n++)
      {
      %>
            <td width="<%= colWidth %>%"></td>
      <%
      }
      %>
         </tr>
         <tr>
            <td class'rightAlign'></td>
      <%
      for (n=0; n<arrayHora.GetSize(); n++)
      {
         var horaDs = arrayHora[n].HORA_DS;
      %>
            <th colspan='4' class="horario"><%= horaDs %></th>
      <%
      }
      %>
         </tr>
      <%
      var spanLocal = 4*arrayHora.GetSize() + 1;
      var localizacionDs = "";
      for(arrSala in arraySalaOrder)
      {
         if(localizacionDs != arrSala[2])
         {
            localizacionDs = arrSala[2];
        %>
              <tr>
               <td colspan="<%= spanLocal %>">
               <div class="localizacion">
                  <span><%= localizacionDs %></span>
               </div>
             </td>
            </tr>
        <%
         }
         headerDs = mapSala[arrSala[0]]["SALA_DS"];
         divTitleStr = mapSala[arrSala[0]]["DESCRI"];
         arrayResSala = mapSala[arrSala[0]]["RESERVAS"];
         arrayResSala.Sort(function(x,y)
                           {
                              if(x.RESERVA_INI_DT < y.RESERVA_INI_DT) return
-1;
                              if(x.RESERVA_INI_DT > y.RESERVA_INI_DT) return
1;
                              return 0;
                           });
      %>

         <tr>
            <th><%=         headerDs         %><div         class='popup'><%=
divTitleStr%></div></th>
      <%
         for(objOpRes in arrayResSala)
         {
            resWidth = Round((objOpRes.HORA_FIN_CD - objOpRes.HORA_INI_CD) /
15);
      %>
            <td colspan="<%= resWidth %>">
      <%
            if(empty(objOpRes.RESERVA_ID))
```

```
            {
               var openReserCd = objOpRes.suffix();
               var reservaDs;
               var duracionArg = DURACION_NM;
               if(!Empty(duracionArg))              duracionArg              =
",%s".Format(duracionArg);
               if(objOpRes.SEARCH_MATCH_FL == 0)
               {
                  if(objOpRes.VALID_RES_FL == 0)
                  {
      %>
                  <div    class='salaLibreInvalid    bckgLibreInvalid'><span
class='reservaSlot'> </span></div>
      <%
                  }
                  else
                  {
                     reservaDs = "Reservar %s".Format(objOpRes.HORARIO_DS);
      %>
                     <div                      class="salaLibreInvalidDuration
bckgInvalidDuration">
                        <a     class     =     "salaNoUnderline"     href="#"
onclick="Top.FrameSubmit.iCoreDoITFWithArgs('GLOBALESSL','CORE','ADDRESERVA',
'<%= openReserCd %>'<%=duracionArg%>);">
                           <span class="reservaSlot"><%= reservaDs %></span>
                        </a>
                     </div>
        <%
                  }
               }
               else
               {
                  if(nuevaReserva)
                  {
                     reservaDs = "Libre %s".Format(objOpRes.HORARIO_DS);
      %>

                     <div class='salaLibre bckgLibre'>
                        <a class = 'salaNoUnderline'>
                           <span class='reservaSlot'><%= reservaDs %></span>
                        </a>
                     </div>
      <%
                  }
                  else
                  {
                     reservaDs = "Reservar %s".Format(objOpRes.HORARIO_DS);
      %>
                     <div class='salaLibre bckgLibre'>
                        <a     class     =     'salaNoUnderline'     href='#'
onclick="Top.FrameSubmit.iCoreDoITFWithArgs('GLOBALESSL','CORE','ADDRESERVA',
'<%= openReserCd %>'<%=duracionArg%>);">
                           <span class='reservaSlot'><%= reservaDs %></span>
                        </a>
                     </div>
      <%
                  }
               }
            }
            else
            {
               reservaDs                           =                          "%s
%s".Format(objOpRes.HORARIO_DS,objOpRes.RESERVA_DS);
```

```
            var                popupDs                =                "%s
%s".Format(objOpRes.HORARIO_DS,objOpRes.RESERVA_LISTA_DS);
      %>
               <div class='salaReservada bckgOcupada'>
                  <span class='reservaSlot'><%= reservaDs %></span>
                  <div class='popup'><%= popupDs %></div>
               </div>
      <%
            }
      %>
            </td>
      <%
         }
      %>
         </tr>
      <%
      }
      %>
      </table>

      <table  class='tableLeyenda'  width="99%"  border="0"  cellspacing="0"
cellpadding="0">
         <tr>
            <td width=10%>Leyenda</td>
            <td class='cuadrado'><div class='bckgLibre'> </div></td>
            <td class='cadencia'>Libre</td>
            <td class='cuadrado'><div class='bckgOcupada'> </div></td>
            <td class='cadencia'>Ocupado</td>
            <td                                    class='cuadrado'><div
class='bckgLibreInvalid'> </div></td>
            <td class='cadencia'>Pasado</td>
            <td                                    class='cuadrado'><div
class='bckgInvalidDuration'> </div></td>
            <td class='cadencia'>Duración inferior a la indicada</td>
         </tr>

      </table>@>
      ansHtml = state;
      ansHtml = "<style type='text/css'>" +
                  GlobalesSL.GetDefaultStyleSalas() +
                  GlobalesSL.GetStyleSearchSala() +
                  GlobalesSL.GetStyleLeyenda() +
                  "</style>" +
                  ansHtml;
      return ansHtml;
   }
```

B OpenReservaSlot Build.

Esta función ha sido la más compleja de desarrollar en el proyecto. Construye reservas en función de los parámetros pasados como argumento. En muchas partes de la aplicación se deseaban construir reservas pero cada llamada necesitaba una construcción especial y muy similar al resto, por lo que se decidió la implementación de esta clase auxiliar que contiene esta función y lo que hace es construir reservas acorde a los parámetros especificados.
La funcionalidad de las reservas que construye se explican en la parte de desarrollo.

```
/**Función Build*/
   public function Build(object objRef, data builder)
   {
      var arrayObjects, arrayReserva, arraySala, arrayHora;
      var onlyDispFl, onlyRealFl, forceFulldayFl;
      var reservaDefDt, lastReservaFinDt, corteDt;
      var duracionNm, itemObj;
      if(HasHeritage(GetObjClassId(objRef),"RESERVARECURSL"))
      {
         return BuildFromReservaRecur(objRef,builder);
      }
      else if(HasHeritage(GetObjClassId(objRef),"RESERVASL"))
      {
         return BuildFromReserva(objRef,builder);
      }
      else if(!HasHeritage(GetObjClassId(objRef),"AP_RESERVA"))
      {
         return;
      }
      forceFulldayFl = (objRef.FORCE_FULLDAY_FL == 1);
      onlyDispFl = false;
      onlyRealFl = false;
      if(builder == "BUILD_PDTE_FROM_SESSION_USR") onlyRealFl = true;
      else if(objRef.DISPONIBLE_FL == 1) onlyDispFl = true;
      if(empty(objRef.RESERVA_DT)) reservaDefDt = ResetTime(SysDate());
      else reservaDefDt = ResetTime(objRef.RESERVA_DT);
      corteDt = "";
      if(reservaDefDt == ResetTime(SysDate()))
      {
         arrayHora = new OpenHoraSL(this,"BUILD_FIRST_FROM_SYSDATE");
         if(!arrayHora.IsEmpty())                    corteDt                  =
DateAdd(reservaDefDt,arrayHora[0].HORA_CD,"MM");
      }
      arrayObjects = new Array();
      arraySala = objRef.GetListaSalaSearch(builder);//Ordenados por nombre
      arrayReserva  =  objRef.GetListaReservaSearch(builder);//  Ordenados  por
fecha de inicio
      for(objSala in arraySala)
      {
         lastReservaFinDt = objRef.RESERVA_INI_DT;
         for(objReserva in arrayReserva)
         {
            if(objReserva.SALA_ID == objSala.SALA_ID)
            {
               if(!onlyRealFl)
               {
                  // Libre inicial + entre dos reservas
                  if(objReserva.RESERVA_INI_DT > lastReservaFinDt)
                  {
```

```
                    arrayObjects                                                =
AddItemsLibre(arrayObjects,objSala,lastReservaFinDt,objReserva.RESERVA_INI_DT
,corteDt);
                }
             }
             if(!onlyDispFl || forceFulldayFl)
             {
                if(forceFulldayFl    ||    (objReserva.RESERVA_FIN_DT    >
SysDate()))
                {
                   arrayObjects.Add({"objRes"  :  objReserva,  "iniDt"  :
objReserva.RESERVA_INI_DT, "finDt" : objReserva.RESERVA_FIN_DT});
                }
             }
             lastReservaFinDt = objReserva.RESERVA_FIN_DT;
          }
       }
       // Libre final
       if(!onlyRealFl && (lastReservaFinDt < objRef.RESERVA_FIN_DT))
       {
          arrayObjects                                                =
AddItemsLibre(arrayObjects,objSala,lastReservaFinDt,objRef.RESERVA_FIN_DT,cor
teDt);
       }
    }
    //Si se busca para reservar se tiene en cuenta el filtro de duracion,
en caso contrario se ignora
    if(onlyDispFl && !empty(objRef.DURACION_NM))
    {
       arrayObjects                                                =
ApplyFilterDuracion(arrayObjects,objRef.DURACION_NM,forceFulldayFl);
    }
    if(onlyDispFl && forceFulldayFl)
    {
       arrayObjects                                                =
ApplyFilterSalaDisp(arrayObjects,arraySala.GetSize());
    }
    // Ordenar resultados por Libre / Ocupada, Hora Inicio Asc, Hora Fin
Desc
    arrayObjects.Sort(function(mapX,mapY)
                      {
                         if(!mapX["objRes"] && mapY["objRes"]) return -1;
                         if(mapX["objRes"] && !mapY["objRes"]) return 1;
                         if(mapX["iniDt"] < mapY["iniDt"]) return -1;
                         if(mapX["iniDt"] > mapY["iniDt"]) return 1;
                         if(mapX["finDt"] < mapY["finDt"]) return -1;
                         if(mapX["finDt"] > mapY["finDt"]) return 1;
                         return 0;
                      });
    for(mapObj in arrayObjects)
    {
       if(mapObj["objRes"]) CreateFromReserva(mapObj);
       else CreateLibreFromReservaDate(mapObj,true);
    }
 }
```

C Diagrama SMAPPS